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

April 15, 2019

Trustable oracles towards trustable blockchains

Pedro Duarte da Costa

Mestrado Integrado em Engenharia Informática e Computação

April 15, 2019

Abstract

Resumo

Acknowledgements

Pedro Duarte da Costa

"If I have seen further it is by standing on ye sholders 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 it's early development days and many different proposals are being worked on as to improve it's 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 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 [**?**], 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 an 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 an 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 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 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 on 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

The Ethereum blockchain is designed to be entirely deterministic [**?**], 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 a 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

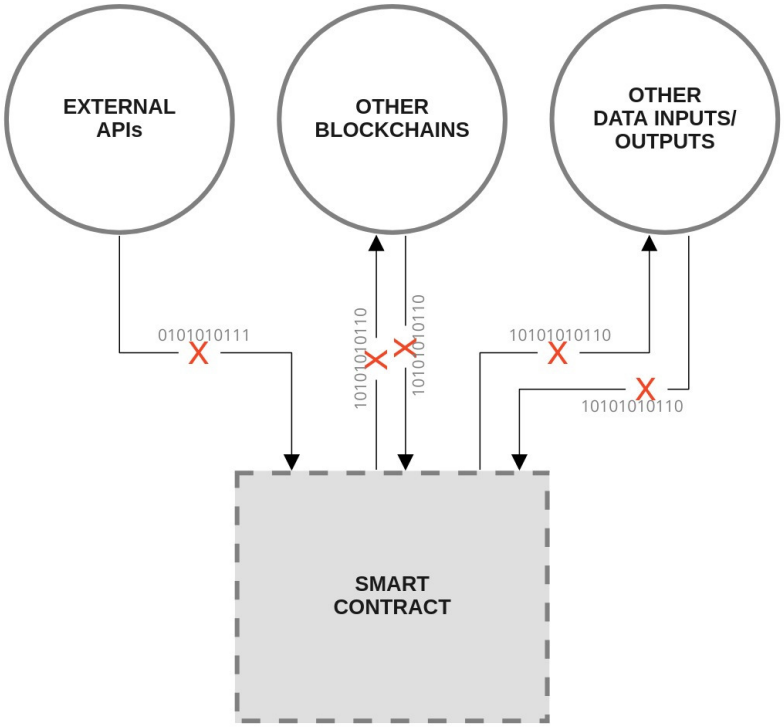

Figure 1.1: Smart contract connectivity problem.

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, 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 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 to 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 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 blokchain is the use of a secure blockchain middle-ware, mentioned before as, an oracle. Oracles can query data from APIs, datafeeds, 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 is ran.

1.6 Oracles Trust

Trust in oracles comprises two main components, service availability, trusting that the service will always return a response to our query, and untampered relay of information, meaning the information the oracle computed or queried is original and not tampered with. If oracles are compromised we risk compromising the trust of the underlying blockchain by inputting falsified information in a system that is trusted to always have a valid state. Therefore it is imperative that oracles provide a proof for the information they provide, as well, as keep a decentralized approach by having a network of oracles always available to answer the queries.

1.7 Trusted Execution Environment

A Trusted Execution Environment (TEE) is a computational environment strongly isolated from the main operating system running on a given device. The isolation is achieved through software and hardware enforced mechanisms. A TEE runs a small-footprint operative system, which exposes a minimal interface to the main operative system running on the device in order to reduce the attack surface. TEEs can run special applications with high-security requirements, such as cryptographic key management, biometric authentication, secure payment processing and DRM. Some

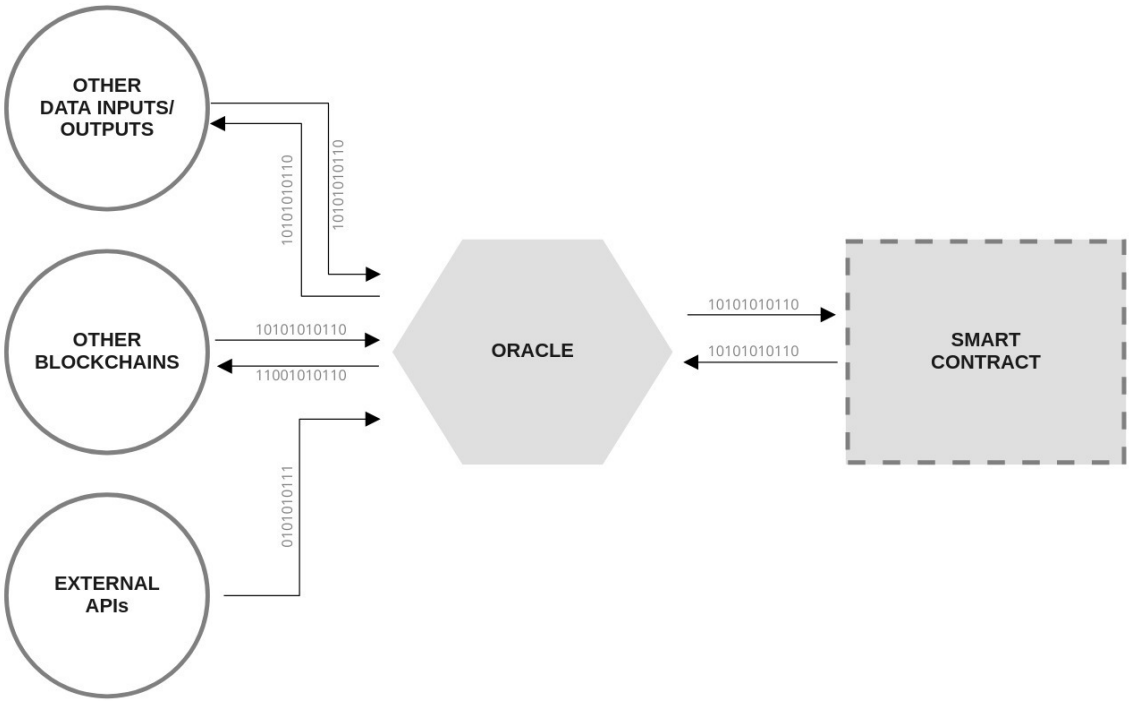

Figure 1.2: Oracle integration.

examples of TEEs are ARM Trustzone-based Secure Elements , widely used in smartphones, and the more recently released Intel Software Guard Extensions (SGX) Enclaves.

1.8 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, who 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. Taikai's project will serve as proof-of-concept for the implementation of trustable oracles.

The growing interest in blockchain technology and specially in the potential of Smart Contracts together with the lack of research on trustable oracles creates a gap on the general adoption of blockchain by business and governments. This gap and the opportunity bestowed with the Taikai project provides the perfect context for pursuing the construction of a bridge to overcome the oracle trust problem and further empower existing and future blockchain projects.

The proposed objectives for this work are as follows:

- Identifying the necessary components for end-to-end reliability between smart contracts and outside blokchain information;
- Providing a general framework for guaranteeing oracle trust;

- Implementing a proof-of-concept in the Taikai project on the EOS blockchain;

1.9 Document Structure

Chapter 2

State of the Art

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 may not yet be documented on peer-reviewed papers but, nonetheless, is invaluable in 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 Research

In terms 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.

2.1.1 Research Questions

First of all and to guide the focus of the research, the following research question was defined:

- RQ1: What kind of blockchain oracles have been proposed?

2.1.2 Review Strategy and Data-sources

Figure 2.1, presents the predefined review strategy used in order to achieve such a goal and maintain unbiased, transparent and reproducible research.

The following 5 electronic databases were used to query for such information:

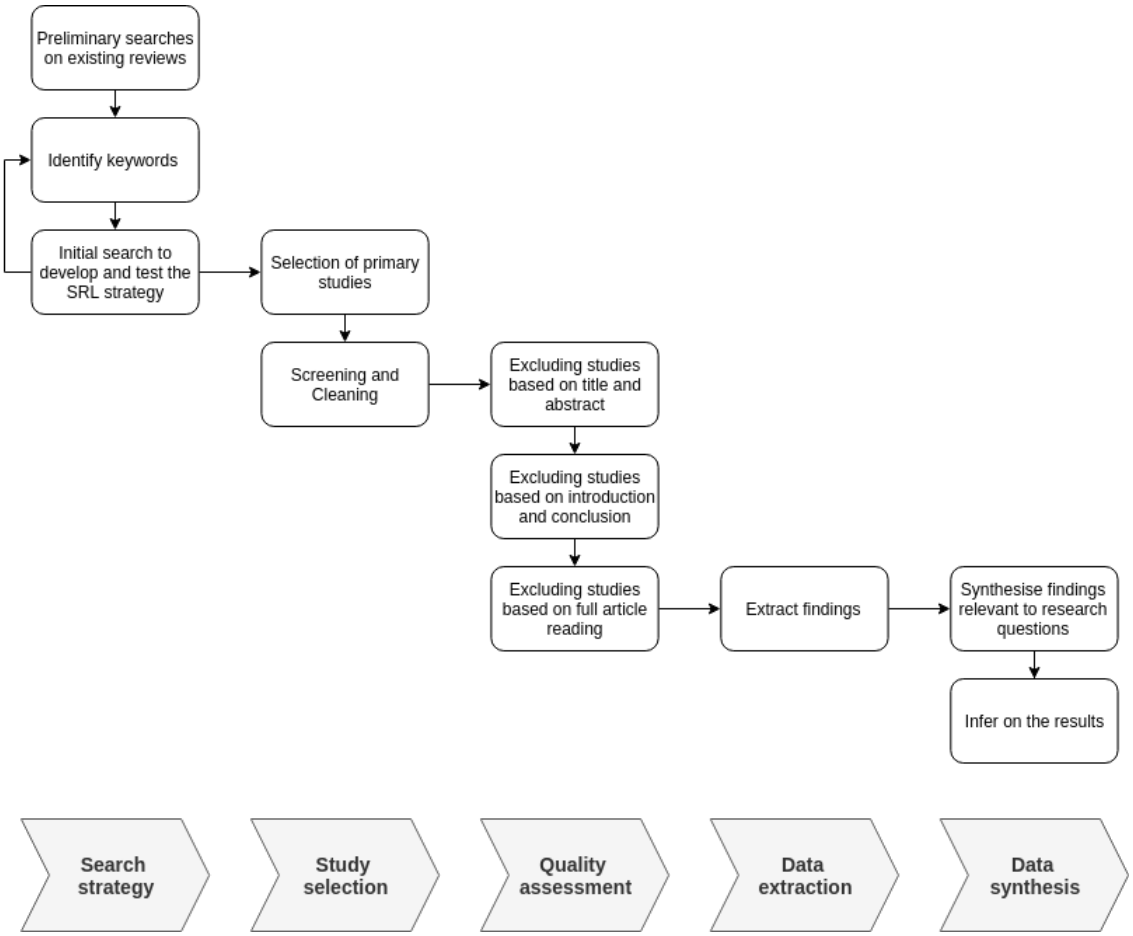

Figure 2.1: Review strategy.

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

The defined search query for the search of the relevant paper 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"))

The search was performed on the 5th of February 2019 and revealed the results presented in 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: Search results 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. Analysing the initial search results per year, in table 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 in the beginning of February.

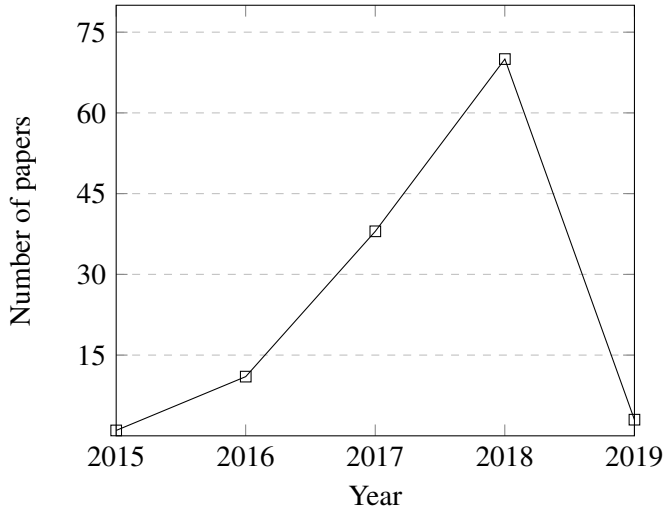

Table 2.2: Search results per year

2.1.3 Study Selection and Quality Assessment

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 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 was finished, only 13 of the 123 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 a implementation of a trustable oracle or an protocol to overcome the trust in oracles.

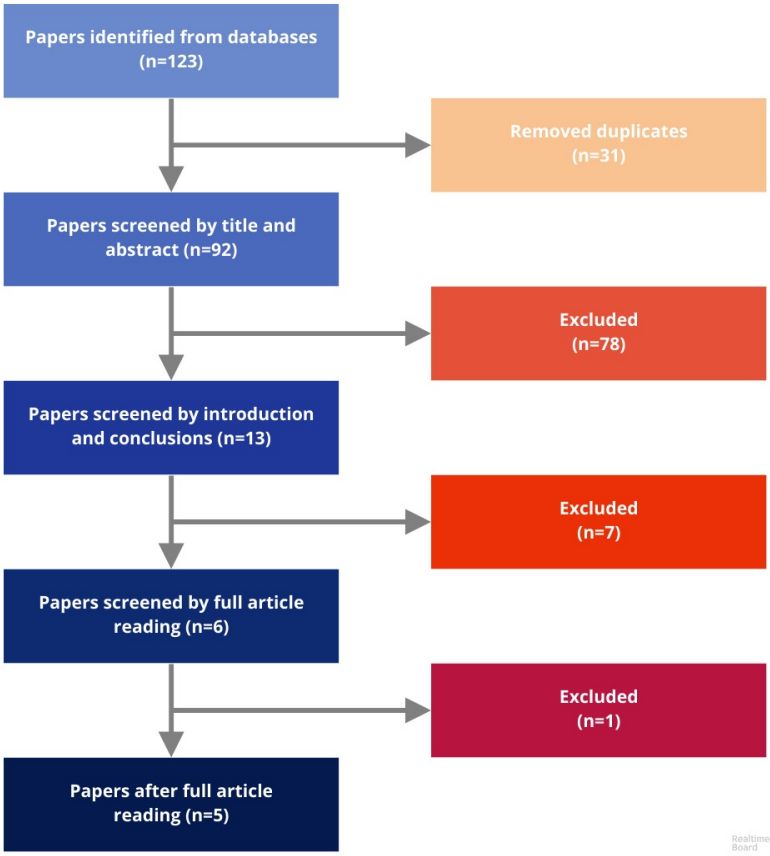

Figure 2.2: Screening stages.

- Stage 4: Full article reading to assess if the final bucket of articles answer the research questions.

2.1.4 Results

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

Town Crier [**?**], leverages trusted hardware, specifically Intel SGX, to scrape HTTPS-enabled websites and serve source-authenticated data to relaying smart contracts. TC architecture involves a TC contract on the blockchain that receives datagram requests from a User Contract on the blockchain and communicates those request to a TC server which then retrieves an answer from a data source through an HTTPS connection.

Astraea [**?**], 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 game theory incentive structure as a means to keep the players honest.

[**?**], proposes a protocol for oracle sensor data authenticity and integrity to an 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.

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

[**?**] does not propose a specific method but analyses existing solutions and defines a systematic classification for existing trustable off-chain computation oracles. The authors identifies 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, which proposes a privacy-preserving decentralized computation platform based on sMPCs where a blockchain stores a publicly verifiable audit trail. However, current sMPC protocols add too much overhead for such a network to be practical. Hence, Enigma now relies on Trusted Execution Environments

- *Enclave-based Computation*, relying on Trusted Execution Environments (TEE) to execute computations off-chain. Identified existing solutions are Enigma and Ekiden which present two different implementations of EOCs. In Enigma programs can either be executed on-chain or in enclaves that are distributed across a separate off-chain network. An Engima-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, 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 Non-Academia Research

To search for non-academia research 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.

2.2.1 Findings

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

- Oraclize.it [**?**], 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[**?**], 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 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, introduces a system of solvers and verifier. Solvers are compensated for performing computation and verifiers are compensated for detecting errors in solutions submitted by solvers.

2.3 Summary and Conclusions

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 on the results is described on table 2.3.

Author / Project	Type	Distributed Network	Achieves trust through
Town Crier	Data carrier	No	Trusted hardware signed attestations
Astraea	Data carrier	Yes	Game theory incentives
[**?**]	Computation	Yes	Sets of public and private keys
[**?**]	Data carrier	Yes	Incentive based
TrueBit	Computation	Yes	Incentive based
Oraclize.it	Data carrier	No	TLSNotary
ChainLink	Data carrier	Yes	Query multiple sources
SchellingCoin	Computation	Yes	Incentive based

Table 2.3: Summary of oracle projects/research.

2.3.1 Conclusions

Two main conclusions come from both the academia and non-academia research.

The first of all, there is a clear lack of academia research on the topic of creating trustable oracles. Town Crier proposes a solution for relying data securely but requiring specific hardware. Astraea and [**?**] uses incentives and game theory as means for good oracle behaviour but doesn't not provide complete trust in edge cases in which pursing erratic behaviour may be worth it. [**?**] is very promising in the field of Internet of Things for oracle sensor authentication but can only guarantee that data was generated by a specific sensor but the approach cannot be generalised for other oracle scenarios.

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.

The literature review points out the lack of research done by the academia in trying to solve one of the most important motives for blockchain general adoption. Only second, maybe, to scalability. The oracle trust problem, efficiently solved, opens doors to the contracts of the futures. Start-ups and sole developers are for now the main force in solving this problem which launches the challenge and motivation for the next chapter of this research.

Chapter 3

Trustable Oracles

3.1 Defining Trust

At this point, a definition of what trust in a 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 favorable 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 signatures 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 most 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 loosing 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 audit. 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.

3.2 Oracle Architectures

3.2.1 Oracle as a Service w/ Single Data Feed.

3.2.1.1 Context

Smart contracts will provably power a decentralized world of automation and trust-less commitments. Companies, groups and individuals will be able to automate tasks and contracts but as far as the ecosystem is, at the moment, smart contracts are 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.

3.2.1.2 Example

3.2.1.3 Problem

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 surge to empower business and smart contract capabilities, connecting smart contracts to the world outside of the blockchain. The problem here is to trust in 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 mechanisms and therefore must provide some kind of proof of honesty.

3.2.1.4 Forces

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

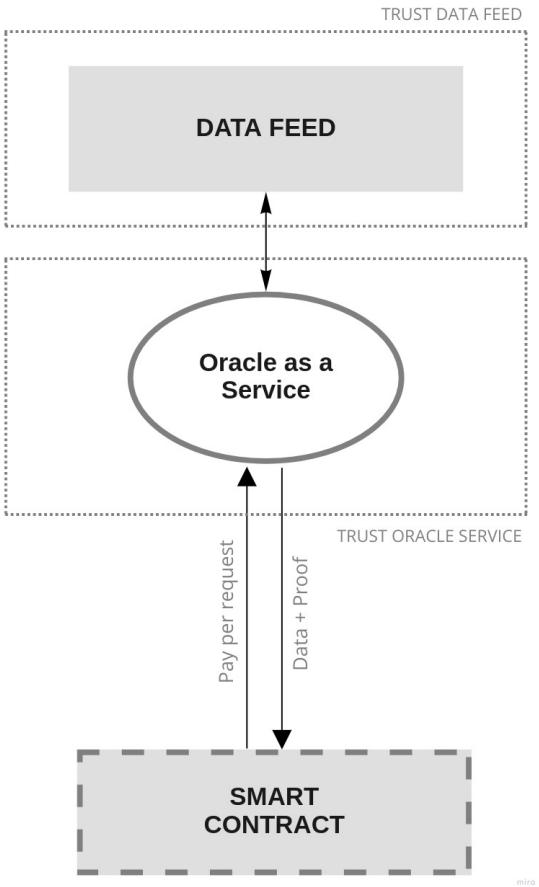

Figure 3.1: Oracle as a Service w/ Single Data Feed.

- **Blockchain Interoperability** - The ability to get information from other blockchains if possible using an oracle that queries one blockchain and inputs the information on another.

- **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.

3.2.1.5 Solution

A solution is to use a secure authenticated channel between any external data-source and blockchain applications, known as Oracle. A proposed and existing solution in the market is Oraclize [**?**]. Currently providing fee-based oracles in which smart contracts pay for each request. In order to achieve a higher level of trust, for an extra fee, it can also provide proofs that the provided data as not been tampered by their oracle. The extent to which this proofs can be trusted will be further analysed.

3.2.1.6 Example Resolved

3.2.1.7 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 behavior, 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 behavior 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 behavior. 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 behavior. By paying the oracle for each request, that becomes the oracle service business model and therefore an extensive record of good behavior if 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 trustees 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.

3.2.1.8 Known Uses

3.2.2 Oracle as a Service w/ Multiple Data Feeds.

3.2.2.1 Context

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.

3.2.2.2 Example

3.2.2.3 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

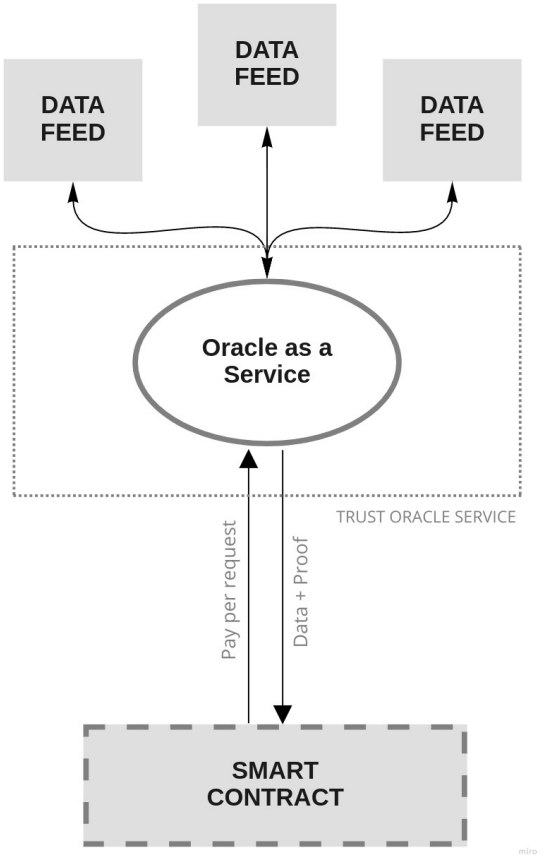

Figure 3.2: Oracle as a Service w/ Multiple Data Feeds.

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.

3.2.2.4 Forces

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

- **Trusting data** - By querying several data sources there is a higher trust on the veracity of the data.

3.2.2.5 Solution

3.2.2.6 Example Resolved

3.2.2.7 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.

3.2.2.8 Known Uses

3.2.3 Single-Party Self Hosted Oracle.

3.2.3.1 Context

Although the use of Oracles as a Service allows for a low 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. 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 behavior.

3.2.3.2 Example

3.2.3.3 Problem

Currently, oracle behavior 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.

3.2.3.4 Forces

- **Higher trust on oracle behavior** - Oracle good behavior is usually backed by authenticity proofs which are expensive to check on chain or don't bring much value to the contract

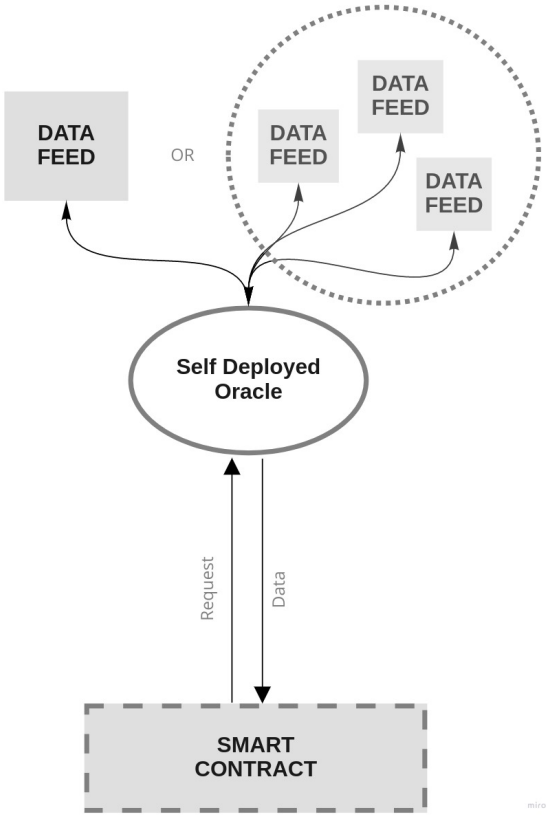

Figure 3.3: Single-Party Self Hosted Oracle.

when verified off-chain since the current contract already executed its code with tampered data.

- **Lower smart contract costs** - Checking authenticity proofs leads to higher contract deployment costs, as the proof can be long and computational expensive.
- **Lower smart contract complexity** - Verifying authenticity proofs on chain requires the developer to have sufficient knowledge to write the verifying functions.

3.2.3.5 Solution

A solution to trusting a 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 behavior, as the developer knows exactly what the oracle is running under the hood.

3.2.3.6 Example Resolved

3.2.3.7 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.

3.2.3.8 Known Uses

3.2.4 Multi-Party Self Hosted Oracle.

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

3.2.4.2 Example

3.2.4.3 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. In terms of availability, a single oracle creates a single point of failure in case of an attack or system failure.

3.2.4.4 Forces

- **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.

3.2.4.5 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

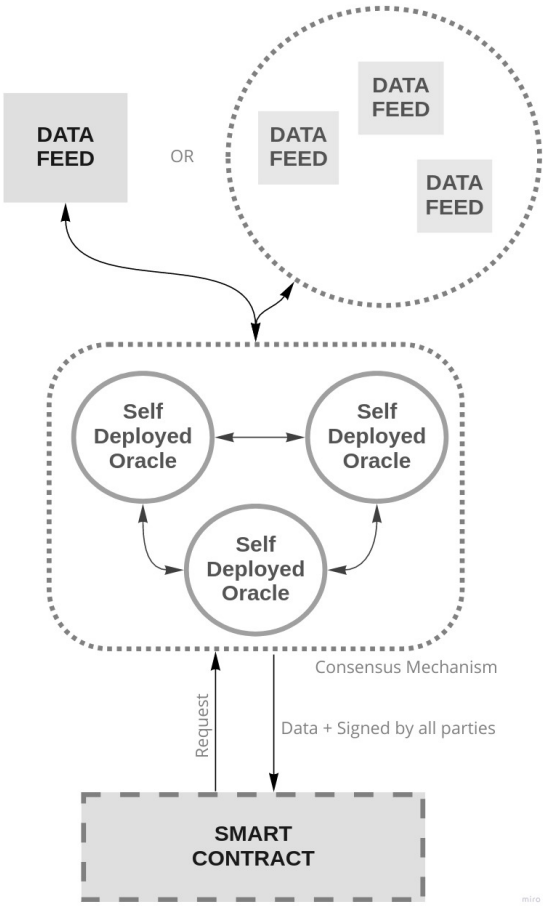

Figure 3.4: Multi-Party Self Hosted Oracle.

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.

3.2.4.6 Example Resolved

3.2.4.7 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 having the same weight in providing the data. This context, however, is only suitable for previously defined user groups, with and 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 [**?**]. 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. // Talvez valha a pena continuar esta parte numa nova architectura em que por ventura se use crowd em conjunto com provas criptograficas.

3.2.4.8 Known Uses

3.3 Summary and Conclusions

Chapter 4

Authenticity Proofs

As technology evolves and becomes mainstream so does the amount of generated data and its distribution. And so it becomes crucial to be able to authenticate a piece of information as originating from a known, trusted source.

In the context of today's web we are accustomed to trust that a certain website or data is originated from the expected source due to the general adoption of the HTTPS protocol. An extension fo the HTTP protocol which creates and 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 not 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 have not direct way of communicating with HTTPS enabled services and therefore obtain authenticated data. Creating a need for a trusted service, which if centralized if fallible unless it can provide an irrefutable proof of its honesty.

Several authenticity mechanisms have therefore been developed and, as seen 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. In this section, I will deep dive on the most common proofs and discuss their implementation and applicability.

4.1 TLSNotary

TLSNotary is a mechanism for independently audited HTTPS 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 handshake protocol to create a irrefutable proof as long as the auditor trusts the server's public key by splitting the TLS master key 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 a 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 a certain traffic did occurred.

4.1.1 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.

$$PRF(secret, label, seed) = PMD5(S1, label + seed) \otimes PSHA - 1(S2, label + seed) \tag{4.1}$$

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

The auditee applies P_MD5 to S1, generating 48 bytes:

$$H_1 = H_{1,1} \parallel H_{1,2} \tag{4.2}$$

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

$$H_2 = H_{2,1} \parallel H_{2,2} \tag{4.3}$$

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

$$M_2 = H_{1,2} \parallel H_{2,2} \tag{4.4}$$

$$M_1 = H_{2,1} \parallel H_{1,1} \tag{4.5}$$

The auditee and auditor calculate X and Y, respectively.

$$X = P\_MD5(M_1) \tag{4.6}$$

$$Y = P\_SHA - 1(M_2) \tag{4.7}$$

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 is 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 that allow him to calculate the server mac key and safely execute a normal TLS decryption and authentication step.

These complex sequence of calculations prevent 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 [**?**].

4.1.2 Limitations

TLSNotary, provides some capabilities to attest TLS connections but comes with several limitation. 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 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 [**?**], 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 trust to a bigger another central entity. It also only allows the existence of one auditor in which we must trust, in a situation in which more than one auditor is required TLSNotary will not be able to satisfy such requirement.

4.1.3 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.

4.2 Android Proof

In the oracle context, the Android proof 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.

4.2.1 SafetyNet Attestation

SafetyNet, developed by Google, is an api service that allows to assess 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 SafeyNet API it allows to assess if the application running on the device has not been tampered with by comparing the application SHA-256 hash with its public available and distributed open-source code.

4.2.2 Android Hardware Attestation

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

4.2.3 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 and

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 used 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 not alteration has occurred on the HTTP response before it is 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 signing process using the initially created and attested key have 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.

4.2.4 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 of this approach, but nonetheless, as long as Google is trustworthy it is a very secure model.

4.3 Ledger Proof

The ledger proof is based on the use of a specific trusted environment, the Ledger Nano S and Ledger Blue, invented by a French company to secure crypto assets safely. This devices implement several layers of hardware and software to prove the security of its execution. These devices run a

specific software, BOLOS, which has an SDK that enables developers to build application which can be installed on the hardware. BOLOS exposes a set of kernel-level API which allow 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 provide 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.

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

4.4 TLS-N

TLS-N [RWG+17], is the first privacy preserving TLS extension that is efficient and most importantly provides non-repudiation. 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 it's 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.

4.4.1 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 an hash value incorporating all previous records, an ordering vector and the time stamp from the start of the session.

The protocol ends when the requester sends a request for evidence. The evidence is composed of a windows of the exchanged ordered messages signed by the generator. The window begins right after the handshake, this prevents Content Omission Attacks, in which if the evidence collection only starts after que request is done, and another request is asked right after (this one, inside the window collection) and the response for the first request is assumed to be the response to the

second one and only this to messages are stored in the evidence window even though the answer was not to the question in the window.

To generate a small proof independently of the number os messages, TLS-N uses Merkle Trees 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.

4.4.2 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.

4.5 Summary and conclusions

Chapter 5

Conclusions and Future Work

The oracle trust problem, is still rather recent, having emerged in 2015 with the deployment of smart contracts on the Ethereum blockchain. The systematic literature review, as far as it has been performed, and the non-academia research review reveal that most of the research and development is being done by the growing and excited blockchain community. Mostly by startups and single interested researchers.

Solving the trust problem and creating a standard for a secure middle-ware between blockchains and outside world information and application provides limitless range of future applications in terms of contracts. Creating, therefore, the grounds and the motivation for the work that will be developed on this thesis.

I hope that the work that will be developed in investigating the necessary requirements and research how oracle trust can be achieved allied with a proof-of-concept implementation on the Taikai projects will solidify the academia position on newly and ground breaking technologies such as the blockchain.

References

[RWG+17] 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.

REFERENCES

Appendix A