Algoracle

Bringing Oracle Technology to Algorand

Abstract: We propose building a decentralized Oracle network on the Algorand blockchain. This network will allow users to run their own nodes, provide their own feeds through an API endpoint, and access a wide range of data feeds for their smart contracts.

Version: 0.2

This draft of the whitepaper is a living document that will be frequently updated over the next few weeks. As new versions are released, some details regarding system architecture may change (navigate the the final page to see updates coming in the next version).

If you have signed up via the website to get this whitepaper, you will receive updated copies to your inbox with each new version, unless you unsubscribe.

If you have any questions, feel free to reach out to the team at:
whitepaper@algoracle.ai

Table of Contents

Executive Summary

The Algorand blockchain is widely considered among the best blockchains in terms of scalability, security, and speed. With its low network costs and quick transaction times, Algorand is one of the most appealing blockchain for new decentralized Applications. However, to support these new application developments, it is essential for the blockchain to have at least one or more robust, secure, and fast Oracle services to provide outside data to Algorand smart contracts.

Algoracle proposes the building of such an Oracle service, and this whitepaper lays out the structure of the application, and the economics and architecture that will be used to both secure the service and maintain appropriate incentives to power the application. Algoracle works by having distributed nodes call API endpoints submitted by Feed Providers. These distributed nodes may then aggregate the values provided by the endpoints, and place the aggregated values (or raw values, depending on the feed) onto a smart contract, where Algorand applications can subscribe to the feeds for a fee.

This whitepaper is composed of two parts. In the first part, a high-level overview of the network architecture, its tokenomics and governance is provided. This part is aimed at non-technical users and primes technical users for Part II. In the second part, a technical analysis is provided into how Oracles may be secured, and how Algoracle plans to do so.

Introduction

Algorand is a high quality blockchain that is very attractive to app developers, especially compared to its largest competitor, Ethereum, which has high transaction fees and low transaction speed. As nearly every industry and hundreds of millions of people adopt blockchain technology in the coming decade[1], we believe the first mover advantage will quickly be reduced once people become more experienced using different wallets for different use cases. With corporations and government entities preferring to build on Algorand, we are already seeing greater adoption among technical users[2][3], and we expect a significant uptick in Algorand based dApps once a functioning and secure oracle has been implemented (considering the amount of capital being committed to developing to the ecosystem[4][5]).

The Oracle Problem

The oracle problem, at its core, means having to trust an entity in a world that should be considered trust less. It could be argued that the only real solution to this problem would be to conduct all activities that an oracle provides onto the blockchain. For example, if ALGOs were only ever exchanged for USDC on-chain, and USDC was only ever spent by individuals on chain, the exchange price of USDC/Algo may be determined on the blockchain, hence solving the oracle problem. However, this preferred state is nowhere close to being realized. Furthermore, basketball cannot be played on the blockchain, nor does the blockchain have a weather system, therefore some external data may never be able to be brought on chain naturally, and instead rely on external sensors for that initial input.

One possible solution might be having everyone watching the game input the score on the blockchain, but what if the loser of a match has more fans, and feel like they deserved a penalty? With the advent of social media, coordinated social 'spamming' a system can and does go viral such as the 'Bum rush the charts' campaign to influence iTunes charts[6], or meme-fueled GameStop buying frenzy meant mainly to bankrupt large hedge funds[7].

By such definitions, the oracle problem may be considered unsolvable. While a detailed analysis of the oracle problem is outside the scope of this whitepaper, an alternative to such a strict definition is that as long as data is sourced from multiple reliable sources, and poor quality or malicious participants stand to lose much more than they gain (in addition to having some

---

[1] Global Blockchain Business Council. (2020). Chain reaction: Blockchain enters the mainstream. Chain Reaction: Blockchain Enters Mainstream. Retrieved November 1, 2021, from https://www.lw.com/thoughtLeadership/gbbc-report-blockchain-enters-mainstream.

[2] https://www.algorand.com/empowering-national-initiatives

[3] https://www.algorand.com/ecosystem/use-cases

[4] https://cointelegraph.com/news/borderless-capital-launches-half-billion-dollar-fund-for-algorand-projects

[5] https://cointelegraph.com/news/former-citi-banker-launches-1-5b-crypto-fund-taps-algorand-as-first-partner

[6] Gilliatt, N., & Gilliatt, N. (2007, March 21). Distributed viral social media spam. Social Media Today. https://www.socialmediatoday.com/content/distributed-viral-social-media-spam

[7] Darbyshire, M. (2021, October 18). Almost 900,000 accounts traded GameStop at peak of meme stock craze. Financial Times. https://www.ft.com/content/df758a2a-6caf-4d5f-ab70-bb5815922b91

barrier to entry), the system can remain secure. In fact, almost all major blockchains are designed as such.

Design Considerations

Layer 1 vs Layer 2

Oracle services tend to generally build out co-chains or blockchains to be able to handle requests, and then interface onto one or more blockchains. While this method allows for growth beyond a single blockchain, the technical infrastructure required is greater than building specifically for a single blockchain. Furthermore, once implemented, it becomes rigid and hard to change (since major changes in a blockchain typically require a fork).

When designing Algoracle, the choice was made to focus on building for the Algorand network. Therefore, Algoracle may be categorized as a decentralized application, although it has a distributed set of nodes (so in reality, somewhere in between). This means a much faster time to market as the only necessary development is the Algorand smart contracts, and the implementation of Algorand's open-source consensus protocols (more on this later). This also means greater focus on security by reducing the number of attack vectors, since there is less code to audit, and tried and tested code is being used for the nodes.

This does not truly limit Algoracle, as the research that has been done as part of building the service has provided great insight into what makes a successful oracle, and the distributed nodes that feed Algorand network can be made to interface onto other blockchains if necessary, in the future. In fact, the name Algoracle is not derived simply by being an oracle for the Algorand network, but rather the fact that the design of the node's consensus mechanism is inspired by the architecture of Algorand; namely, how nodes are selected to propose the next value to put on the smart contract, and how that value is voted on and certified by other nodes in extremely quick timelines.

True Decentralization

Another consideration in the level of decentralization (aka permission to participate). Although Algoracle is not in and of itself a blockchain, it is very similar in that it is a distributed system of nodes and data providers, working together to affect the blockchain state - and as such, a level of permission, if any, needs to be determined.

On one spectrum, a permissioned Oracle service can require knowledge the feed provider is, and only a select number of these known feed providers to submit feeds. On the other end of the spectrum, a permissionless service will neither require nor care who signs up to provide feeds or run nodes.

In their article "When Permissioned Blockchains Deliver More Decentralization Than Permissionless" (Bakos et al.)[8], the authors make a compelling argument that "while distributed architectures may

---

[8] Bakos, Yannis and Halaburda, Hanna and Mueller-Bloch, Christoph, When Permissioned Blockchains Deliver More Decentralization Than Permissionless (September 25, 2019). Communications of the ACM, Available at SSRN: https://ssrn.com/abstract=3715596 or http://dx.doi.org/10.2139/ssrn.3715596

enable open access and decentralized control, they do not preordain these outcomes…permissionless access may result in essentially centralized control, while permissioned systems may be able to better support decentralized control." What this means is just because a system is built to be decentralized and distributed, doesn't mean it will be, largely due to malicious forces or large actors with economies of scale taking over. For example, in the case of *the DAO*, a project build on Ethereum that allowed anyone to participate, hackers were able to call code open to anyone and drained funds from a wallet. As for economies of scale, how bitcoin miners concentrated in a specific region dominated the hash rate (although this has since been improved).

The Algoracle team believes that decentralizing as much as possible is essential but agrees with the authors above that some form of permission is necessary. This is implemented via a deposit mechanism, where Feed Providers and Node runners must stake and deposit a certain amount of network tokens to participate. This raises the barrier to entry, with the goal being to make being a malicious actor or a poor-quality provider as unattractive as possible.

However, while this model will work for majority of feeds, Algoracle also need to consider the fact that not all data providers are created equal, and the requirements of certain applications require no compromise in speed and security, while other applications require access to data that is exclusive. While all feeds will have the same architecture, Algoracle also proposes the creation of 'Institutional grade feeds. These feeds have higher subscription costs which go to fund institutional grade data providers, in addition to higher requirements to run nodes. These feeds may also choose to restrict feeds to certain customer based on pre-existing conditions.

Part I

In the first part, a high-level overview of the network architecture, its tokenomics and governance is provided. This part is aimed at non-technical users and sets the overview for technical users for Part II.

Algoracle Overview

Algoracle hyper focuses on keeping accurate and up to date real-world data on a smart contract with as little infrastructure as possible (enough to move data and secure the system). To ensure the data is fully decentralized, Algoracle keeps the collection and aggregation mechanisms as decentralized as possible. To better explain how Algoracle's proposed architecture will work, we start with the different stakeholders of the system, and its components.

System Stakeholders

Feed providers (FPs)

Feed Providers (FPs) are the backbone of the Algoracle system. They provide the data, sourced from any ideally trusted sources off chain. Algoracle makes it as easy as possible for FPs to contribute to the system. They simply need to input their API endpoint and respond to verified nodes with their values (Vs). There are two types of FPs: Institutional grade FPs and FPs (more details on this distinction to come).The main difference is that Institutional grade FPs generally exercise greater control on the requirements to read from their data sources, and whether that data is aggregated.

Node runners (NRs)

Node runners (NRs) call the APIs provided by the feed providers. Using a cryptographic sortition protocol, the cluster of nodes selects members of the node to propose, vote on and certify the next value(s) to be added to the feed smart contract (FSC). NRs can either aggregate the data or provide the raw values for the smart contracts reading the data to aggregate themselves. There are two types of NRs: Institutional Grade NRs and NRs (more details on this distinction to come). Institutional Grade NRs typically meet greater requirements such as having a licence to read from Institutional grade FPs

Feed Consumers (FC)

Feed consumers (FCs) are the smart contracts that will be using the Vs submitted by FPs. FCs simply call the FSC in the time interval they wish, pay the required fee, and get the data from the contract's global state. Depending on the data feed, FCs fees may come in the form of periodic subscriptions, or charge per requests. Some feeds are open and can be read from the global state of the feed contract.

Algoracle Community (AC)

The public (AC) can view live feeds and report poor quality or malicious FPs to get rewards. They can also vote on previous reports, and once certain thresholds are met, individuals who voted and reported get paid bounties for correct reports (or lose their stake for incorrect reports).

System Components

The following system components are all decentralized and on either the blockchain or a P2P database..

Algoracle website

The Algoracle website was built with react and hosted on a P2P hosting service. It is a front-end website with the backend being either an Algorand smart contract or a P2P database. ACs can vote/report feeds through here and see historical feed Vs. The front-end is decentralized using Skynet's Homescreen, which means the front end can be loaded from users' storage in addition to the P2P option.

Algoracle Nodes

Algoracle nodes are software scripts that run pre-specified code. This code adds the NRs to clusters, and calls the FP's APIs in order to aggregate the data (or pass through the raw data). Once the data is ready, the node software uses the same cryptographic sortition protocol that the Algorand blockchain uses[9] to select which nodes will "propose" the next value to add to the relevant data feed. The protocol also selects voters to vote on proposed value, and certifiers to confirm the vote. Nodes also perform validation checks to ensure valid messages are transmitted through the network.

Nodes will continually pull data from the APIs every few minutes. The specific time is different for each feed; crypto currencies may refresh every 10 seconds, while event dates may only need to be updated once per hour. A historical analysis of each proposed feed is done to determine how often feeds should be polled.

Feed Smart Contract (FSCs)

Feed smart contracts are collectively the contracts that handle the Algoracle ecosystem on chain. They consist of:

a) Index Contracts: store the addresses of the various available feeds.
b) Reputation Contracts: stores the nodes who have staked a deposit to run nodes (it does not track reputation in the traditional sense, but rather its reputation can be discerned from its history of penalties/rewards).

[9] Algorand's Cyrptographic sortition algorithm- https://www.youtube.com/watch?v=XfP862hCrDM

c) Payment contracts: pays out rewards, collects penalties, and handles staked deposits and withdrawals and fees.
d) Data contract: where the actual feed data is stored.

P2P database (P2PDB)

Algoracle uses a P2P database to store historical information about the data submitted by FPs and NRs.

Table 1 The different stakeholders of the system and the components they generally interact with.

	Website	FSC	P2PDB	Algoracle nodes
Feed Providers	•	•	•	
Node Runners		•		•
Feed Contracts		•		
Algoracle Community	•	•	•	

Algoracle System Life cycle

FP add data feed

FPs signs up to provide data feeds via the Algoracle website and pay a deposit. The deposit is handled by the payment smart contract, and once confirmed, the API of the provider is added to the indexing list. The FP also would need to setup a webhook or script that will be used to determine the list of authorized NRs, then store the public keys of these nodes, as these nodes will use their digital signatures when calling the API. Permissioned feed providers may limit consumers and node runners based on some pre-existing conditions.

NRs sign up to become node runners

NRs are required to sign up and pay a deposit to join the node cluster and begin pulling Data. NRs communicate via gossip protocols, which are fast, and use extremely quick algorithms to determine the proposers, voters, and certifiers. The message sizes between nodes are kept low, and messages from unauthorized or malicious nodes are discarded rather than propagated. With the simplicity of running a node, we except thousands of nodes to operational per feed, and a limit of 4,096 feeds may be set. Permissioned feed providers that guarantee a certain quality of their feed would naturally limit the number of NRs to a likely maximum of around 30. These permissioned FPs may require that these NRs purchase licenses to further restrict access and add another layer of security to protect the exclusivity of their data.

NRs pull data from FPs

With the API information in place, NRs can begin calling the API endpoints. Nodes also communicate with each other to determine the proposers, certifiers, and voters. Then a set of authorized nodes will sign the transactions together to put the data on chain, and the payment contract pays out Algoracle tokens to honest participants and penalizes poor quality/malicious actors, respectively. NRs are considered poor or malicious if they execute code that is not recognized. The Data from non-permissioned FPs may also be stored on a P2P database.

AC participate in bounty hunting

The AC can view the live data or the historical data submissions and stake their tokens to report obviously incorrect feeds. This program is simply an additional layer of penalties on top of the slashing penalties applied to poor quality or malicious feeds (more details on these in the economic and security section in part II).

Tokenomics

Note: this section is constantly being updated, and we are looking for advisors who have experience in this space to help with the tokenomics design strategy.

Using a native token GORA creates greater incentive for honest and reliable participation. Honest early adopters stand to profit from the appreciation of the token an initial low cost, while the Algoracle system will benefit from having high quality and honest providers to anchor the system. In the economic simulations done in part two, we conclude that having a large group of early honest participants results in a system that gets harder and more expensive to attack. These early adopters need a sufficient promise of ROI for early participation, especially while Algorand dApps are being built to use Algoracle during the first several months after launch.

Implicit Incentives

On top of the explicit incentives for good behavior, nodes will also operate on implicit ones. Good and accurate performance of nodes means they will remain on the network, giving them future economic opportunities. Strong performance by the network creates greater trust in it, which will likely result in greater use, and greater future economic opportunities. Nodes will, as a result, have an incentive for the network to perform well and accurately. Undermining trust in the network would undermine confidence and threaten future use and economic returns in the future. There will also be an incentive to provide affordable prices since lower fees will mean more users on the network. While fees from dApp users will be a source of income, the appreciation of the GORA token will provide additional long-term benefits.

Token Distribution

Under construction

Algoracle is currently working with BrightNode Tokenomics consulting firm to determine the optimal number of tokens to launch with, the initial distribution and other considerations such as whether to have an initial supply or mint based on activities on the network. Algoracle understand the importance of tokenomics to any project, and therefore are interested in getting this right by bringing in consultants with deep expertise in this field.

Governance

This section is under construction

All systems generally need to evolve as the number users, FPs, and NRs increase and their motivations change. As a decentralized system, Algoracle will need a way to allow a way for these changes to happen transparently way, that benefits the stakeholders. The main stake holders, in order of importance, are:

1 - Feed consumers
2 - Feed providers
3 - Node runners

The users who consume the feed the most are the most sensitive to changes. Trusted feed providers and node runners are also affected. The governance is being worked on with consulting firms to determine the most sustainable and fair method.

Part II

This part explores:
The technical architectures of the application – how it will be built
A methodology for creating a self-sustaining economy

Introduction

Oracle services provide blockchains with off-chain data, and in the process introduce an additional vulnerability into the blockchain system. Since any system's security is as good as its weakest link, a well-designed oracle system should not contribute to security flaws. On Algorand, one of the most secure blockchains, a proper design is even more crucial.

As such, the research and methodology used to determine the implementation of security features in this paper assumes that Algoracle will live in harsh, hostile environment, where attackers will persistently try to manipulate the system. In such an environment, the security is guaranteed when most participants are honest and the system has great uptime, while the system should be able to ring the alarm bells and/or shutdown when an attack is successful, and possibly even compensate victims.

There are two main points of attack that need to be guarded against.

A) The data that is provided by Feed Providers (FPs) needs to be valid and correct data.
B) The Nodes must honestly aggregate and report the correct value.

The following research does not assume its 100% possible to build such a permissionless system, but rather sets out to prove whether this is possible, and if so, under what conditions, and what trade-offs (if any) is required considering the byzantine nature of Oracles. And within the context of those findings, to build a secure, fast, and scalable oracle solution.

Economic Simulation

To Solve the oracle problem, smart contracts applications generally depend on data from multiple sources. An Oracle takes data from multiple sources and selects the median value. If all providers were always honest, this method would work great to smooth out variations in differences between sources. However, an oracle system must account for malicious and poor-quality feeds. This section will first provide the background on why this is needed, before describing the methodology used to come up with the simulation algorithm to both:

A - test the hypothesis that it is indeed possible to have a self-sustaining oracle system that can ward off attacks and rid the system of poor-quality feeds over time, and
B – it can provide an incentive great enough for node operators and feed providers to profit from honest contribution to the system.

Background

Let's take the example of the current temperature of a city, for example Toronto. Different data providers may report slightly different values. The table below shows the values at a point in time taken from various endpoints.

Data Provider	Submitted Value
#1	12.3
#2	12.5
#3	12
#4	12
#5	12.5
#6	11.2
#7	11
#8	14
#9	12.3

#10	11.9
#11	12
#12	20
#13	12
#14	13
#15	12.5
#16	11.7
#17	12
#18	1
#19	12.1
#20	10

In the table above the median is 12 degrees, while the average/mean is 11.9. Generally, Oracles take the median value, to prevent outliers from influencing the data heavily.

A smart contract relying on only one of the sources above may have gotten values from #8, #12, #18 or #20, and that data would have been considered incorrect, as these data points are outliers. But by relying on an Oracle that aggregates data from all twenty sources, a smart contract can be as safe as possible.

In the scenario below for sports feeds, the number must be exact, unlike temperature. Feed providers #M & #G have provided a value that doesn't match. Regardless how close the values are to the median; sport scores should have no deviation whatsoever.

Therefore, an acceptable median absolute deviation for sport scores may be 0 goals, while for temperature, it may be 0.4 degrees.

Data Provider	Submitted Home score Value	Submitted Away score Value
#A	2	1
#B	2	1
#C	2	1
#D	2	1
#E	2	1
#F	2	1
#G	2	3
#H	2	1
#J	2	1
#K	2	1
#L	2	1
#M	3	1
#N	2	1
#O	2	1
#P	2	1
#Q	2	1
#R	2	1

#S	2	1
#T	2	1
#U	2	1

In the temperature example table above, feed providers #12 and #18 should be considered as malicious as the data is extremely incorrect. Feed providers #8 and #20 can be considered poor quality sources, but not malicious. The remaining feed providers are considered honest, and good quality. The closer to the median value, the higher the quality of the feed.

A decentralized system does not have a central authority to punish bad actors, nor can it prevent anyone who wants to submit a value to do so. However, there must be a mechanism to punish bad actors, eliminate poor quality sources, and reward the highest quality sources to keep the system self-sustaining.

Oracles can solve this by requiring a deposit of funds to be able to submit feeds. Smart contracts that use these oracles pay a fee to use the values submitted.

· When a bad actor submits a bad value (a value considered to be an extreme outlier), they are penalized with some or all their funds. A feed provider whose funds fall below a threshold is removed from the system.
· When a value submitted is considered poor quality (i.e. outside the acceptable median deviation), the penalty may be soft.
· When a feed provider submits a value that is acceptable, they are rewarded.
· It’s possible a feed provider is not rewarded or penalized for submissions that are within the acceptable MAD but still far from the median.

Research Methodology

Data sources may be malicious, or they may be poor quality. It’s hard to tell from just one submission. However, since data providers submit values every few minutes, using rewards and penalties appropriately can create a picture of the reliability of each feed.

To do so, the following are the inputs that need to be considered:

- Deposit amount
- The number of feeds
- An acceptable median absolute deviation for the submitted values
- The number of simulations (one round of submissions is considered one simulations) to run.
- The Total Value* depending on the data (aka TVL, total value locked).

The goal is (these are not final numbers, and are flexible):

- Feed providers should earn between 10%-50% profit on their deposit annually.
- Poor quality sources should lose 5-15% of their deposits annually.
- Malicious sources should face heavy penalties of up to 100%
- The deposit/reward/penalty system should be self-sustainable after any number of simulations.
- The total value should not be more than 2-3% less

*The total value locked is a unique input, which is subjective, that was introduced. This value represent the most money a single dApp could lose in the event of a successful attack. For example, if the largest dApp using Algoracle stands to lose 1 Million, that number would be used. If the system is secured against the highest loss, it should be secure for everyone else.

This value would go down when the absolute median deviation of the combined feed providers is greater than the acceptable median deviation for that data feed.

For example, if the Total Value Locked depending on the sports scores is $100,000, then every time the group of submissions is highly variable, this number would go down, eventually reaching 0 if the MAD of the data the feed providers submit is consistently higher than the acceptable MAD. The higher the difference, the faster the TVL would go to 0.

Note: while the TVL will be used to determine starting deposits, the current draft of this research omits TVL and will introduce this value in the second draft of this paper in the coming weeks

Research Results

Before looking at the simulation results, we establish a baseline to understand how to best validate incoming data, based on historical data points.

Robustness of the oracle

When data is fed into the oracle by providers - several situations can occur. One can be categorized as the ideal scenario where all providers are honest. This would indicate that the rewards and penalty mechanisms alongside a high but reasonable deposit is excellently incentivizing good behavior. In such a situation, the median is more than robust in providing oracle users with accurate data. With increasing numbers of feed providers, one can expect robust data to consistently be provided even when some feed providers input wrong data.

However as evident by famous cases of DeFi hacks and malicious attacks - such negative behaviors are possible and it is important to be able to categorize them and find solutions to develop a robust Oracle service. We can start to slowly look at the increase of malicious attacks in terms of percentages (i.e. 10%, 20%, etc), and understand the difference between several attacks with exact alignment in values and those without alignment in prices.

The oracle will require tools, or tests, to be able to deal with these situations while also recognizing good data. In the following subsection we will start to understand the impact of malicious inputs on the median, the spread around the median and the characteristics of exchange data which we can categorize as honest feed providers.

How malicious scenarios affect oracle data aggregation

To demonstrate the robustness of the median for most scenarios it will encounter, we conduct a simulation study to understand at what point can the aggregated prices start to become dissimilar to market prices. Below we will see results from such a simulation where values from the uniform distribution replace a random subset of exchanges where each percentage indicates the number of exchanges taking in the simulated value

$$(floor(p * N))$$

This is run for three cryptocurrency coin data - Bitcoin, Litecoin and AAVE. The range of values being simulated differs for bitcoin and the other two where reasonable and intuitive ranges are chosen that can help differentiate the malicious effects without unrealistic extreme outcomes.

$$bitcoinexch_i \sim uniform(10000{,}150000)$$

$$litecoinexch_i \sim uniform(0{,}1000)$$

$$AAVEexch_i \sim uniform(0{,}1000)$$

This will help in simulating an in-discriminant attack where the distribution will choose which side of the price it wishes to attack. This will be an assumption for the research methodology chapter.

The numbers 1-6 below indicate 6 consecutive time points

Bitcoin

```
##   Good_data X10_per_Malicious_data X25_per_Malicious_data
## 1  60597.06               60597.06               60592.44
## 2  60858.67               60858.67               60862.96
## 3  60804.61               60821.68               60829.05
## 4  61026.20               61026.20               61026.20
## 5  61704.20               61704.20               61716.72
## 6  63038.00               63038.00               62989.24
##   X50_per_Malicious_data X90_per_Malicious_data
## 1               60596.27               60575.04
## 2               60861.78              104821.94
## 3               60828.78               91060.16
## 4               61035.64               73215.59
## 5               61704.20               67722.08
## 6               62989.24               62926.35
```

Litecoin

```
##   Good_data X10_per_Malicious_data X25_per_Malicious_data
## 1    189.24                 189.25                 189.29
## 2    190.20                 190.17                 190.24
## 3    189.35                 189.35                 189.35
## 4    189.72                 189.72                 189.72
## 5    190.80                 190.79                 190.80
## 6    191.27                 191.27                 191.30
##   X50_per_Malicious_data X90_per_Malicious_data
## 1                 189.29                 478.32
## 2                 190.24                 364.39
## 3                 189.34                 391.25
## 4                 189.71                 413.90
## 5                 190.82                 464.25
## 6                 191.30                 256.01
```

AAVE

```
##   Good_data X10_per_Malicious_data X25_per_Malicious_data
## 1    309.62                 309.62                 309.62
## 2    312.40                 312.73                 312.73
## 3    310.79                 311.05                 310.72
## 4    310.87                 310.87                 310.87
## 5    312.75                 312.75                 312.75
## 6    315.68                 315.78                 315.65
##   X50_per_Malicious_data X90_per_Malicious_data
## 1                 309.80                 567.85
## 2                 312.77                 311.65
## 3                 311.05                 310.72
## 4                 310.87                 310.87
## 5                 312.75                 675.24
## 6                 315.78                 851.83
```

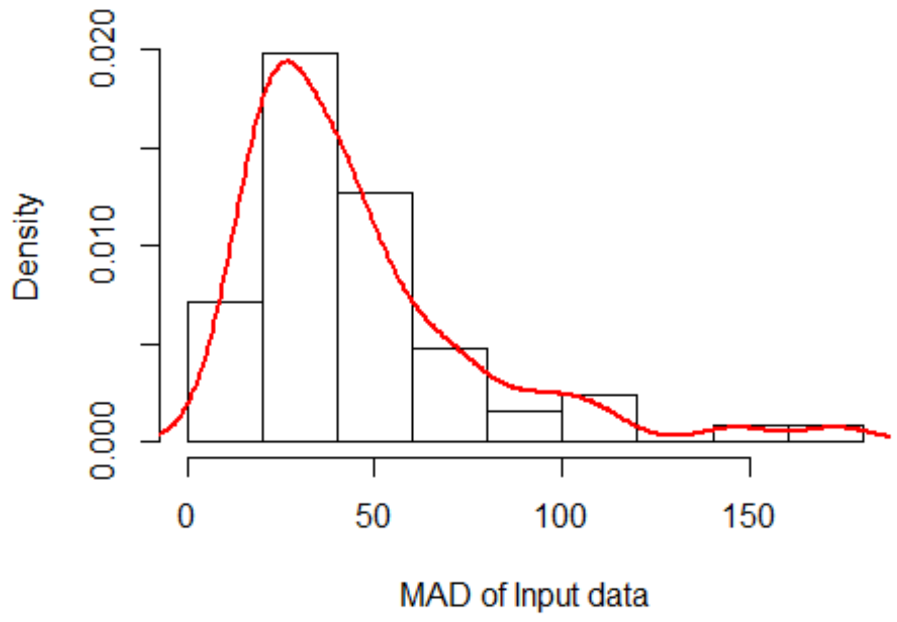
Overall, we see a clear robust outcome from this study where the median can be relied on for the data aggregation process for a wide range of scenarios. A takeover can be considered to be when over 60% of the data are malicious and at this point, we observe significant differences between market prices and the simulated malicious aggregated value. To tackle such a situation, we must begin to understand statistically what characteristics constitutes "Good data".

Characteristics of cryptocurrency feed data

MAD

We will now start to calculate Median Absolute deviance (MAD) of inputs for each time point for Bitcoin data and plot the density plot to observe the distribution of the MAD to understand how the spread around the median changes over a reasonable time period.

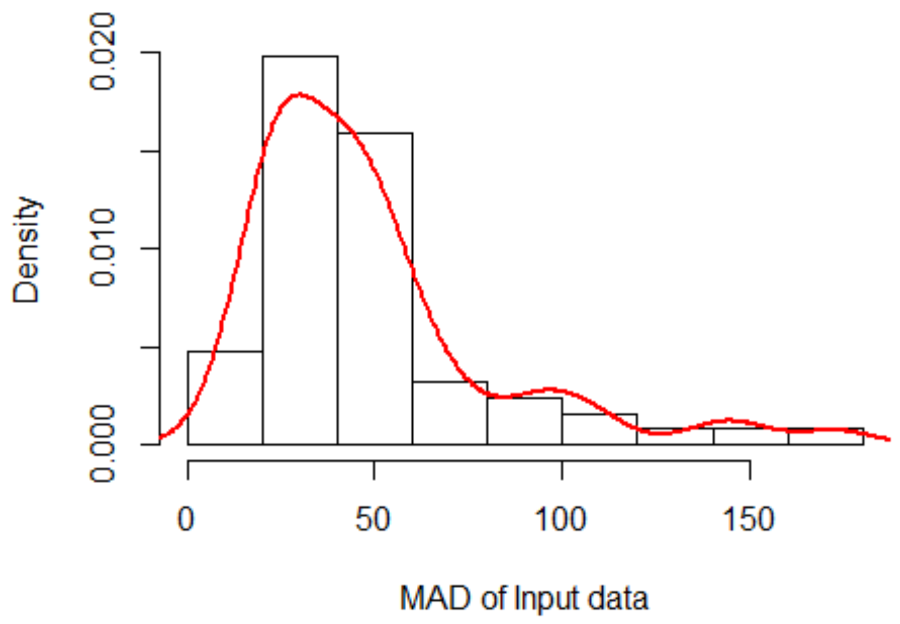
Below we see the spread of the real MAD values for several time points of Bitcoin exchange data.



```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   9.414  23.261  36.816  45.223  54.661 173.030
```

Bitcoin - 10% Malicious providers

What does the distribution look like if three of providers are malicious? For a random subset of 3 providers in the data, we will replace exchange data with a simulated value from the uniform distribution between 10000 and 100000 - all values equally likely to occur.

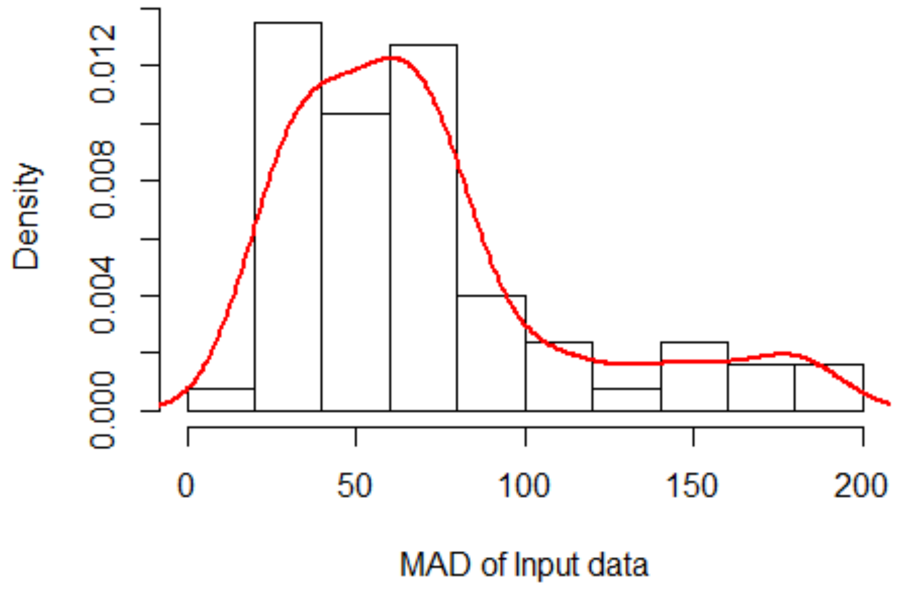


```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   7.834  25.105  40.423  48.448  57.034 173.030
```

The distribution is wider with a longer right tail with much more MAD values possible.

Bitcoin - 25% Malicious providers

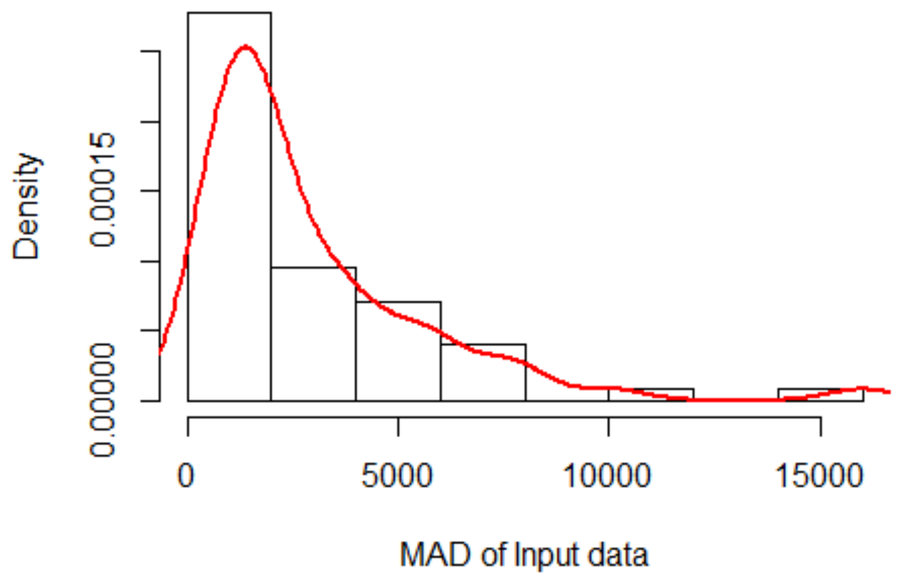
What would the spread of the MAD values look like if we have 25% of the providers as malicious? As before - we simulate values from a uniform distribution and replace the values of the exchanges. In this scenario we have 7 random malicious providers.



```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   7.834  25.105  40.423  48.448  57.034 173.030
```

Bitcoin - 50% Malicious providers

What would the spread of the MAD values look like if we have 50% of the providers as malicious? In this scenario we have 13 random malicious providers.

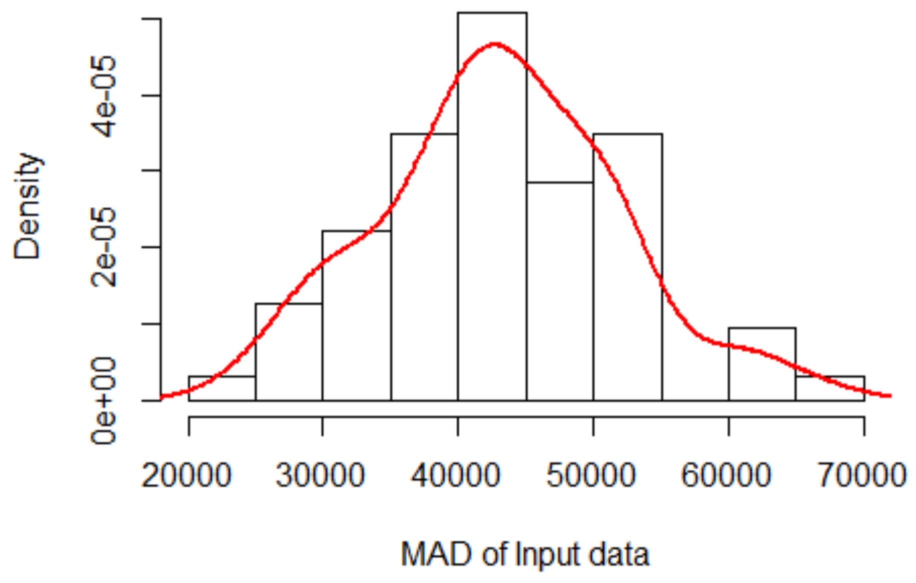


```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   295.5  1135.6  1816.5  2907.2  3848.1 15969.4
```

At 50% malicious providers - the effect is clear to see. Large uncertainty with a distribution that includes extremely large MAD values.

Bitcoin - 90% Malicious providers

We now attempt to modify 23 random providers at each time point into malicious providers.



```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   24610   38224   42676   43111   49652   66184
```

Here the effect is even more defined, we see a rough median of around 3000 with values seen in the best-case scenario extremely unlikely to happen. Unlike the median, the MAD becomes unusable after 25% takeover. Using thresholds based on historic good data however can help detect such behavior as soon as possible keeping the system robust.

CV

When looking at the prices at each time point for several exchanges, a pattern emerges that is specific for different cryptocurrencies. For example - for bitcoin - there is a known large discrepancy between exchanges in terms of prices due to trading volume differences. However, for other cryptocurrencies like AAVE and litecoin, prices are almost identical to each other. An oracle must be able to assess that both types of variation are acceptable. One metric that helps to generalize such a pattern is the coefficient of variation (CV)

$$\frac{sd(x)}{mean(x)}$$

where we look at the spread of the data relative to the average as a percentage. This helps to meaningfully compare variability between coins of different sizes and obtain a generalized threshold that helps the oracle to interrogate data at each time point.

What CV value ranges do we get from the case study datasets? For bitcoin we can see values between 0.04% to 0.41% based on the exchange data and for Litecoin we observe values between 0.01% to 0.44%. Variation within the values at each time point is extremely small and similar. This is a useful fact to leverage against malicious data where we observe large increases in the variability of the data at 90%.

Additional areas to pursue

As part of our research, it is important to utilize the breadth of statistical methods available to help refine the oracle's ability to aggregate good data.

Another way of leveraging relative variation would be to use standardized scores where we scale feed data by centering using the mean and then standardizing so the overall distribution parameters for the data is mean 0 and standard deviation 1. This allows the oracle, in addition to the above methods to understand which feed input is straying away relative to the others. This can help in selecting the appropriate feed data to use as part of the aggregation process.

Other methods under consideration that may have a big impact depending on the computational speed is to use a clustering algorithm such a k-medians with two clusters to help detect portions of bad and good data in situations of malicious attacks. Dependent on possible back testing to test which group contains the good data in addition to the above proposed tests will help the oracle to make good of a bad, but unlikely, situation to its' advantage.

Simulation

Assumptions

1. There's a true underlying value (say a price or a temperature) which takes a random walk with time (minute by minute).
2. For every submission, an individual feed provider generates a submission by sampling from a Gaussian distribution with mean equal to the true underlying value and standard deviation depending on the 'quality' of the feed provider.
3. Feeds providers are of varying 'quality'. A high-quality feed provider has a lower standard deviation and hence is more accurate on average compared to a low 'quality' feed provider. This quality / standard deviation remains constant for each feed provider with time.
4. Every feed provider puts up a certain deposit as collateral. Each minute, the true value takes a random walk, feeds submit values according to the process described above and then these feed providers get rewarded and penalised by the oracle (which is going to be described later).
5. The process above add maliciousness to the feed. To eliminate maliciousness in the feed, set the Acceptable MAD below to 0.

The following parameters are required to run a simulation:

1. nDays = number of days for which one wants to run the simulation (each day consists of 1,440 data calls, i.e. once per minute).
2. TVL = total value locked in the system
3. nFeeds = number of feeds
4. goalReturns = desired annual returns for a perfect feed provider
5. acceptableMAD = acceptable mean absolute deviation
6. nUsers = number of users
7. badK = multiple of acceptableMAD beyond which a submission is considered poor enough to be penalised
8. bigLossK = standard deviation of feed (as a multiple of the acceptableMAD) that should lose 100% of its deposit annually

Algorithm

Every minute, all the feed providers submit data (or are queried for data). The median (M) of all submissions is computed. The fixed reward for a good submission is $R. The fixed penalty for a bad submission is $P. The methodology for calculating R and P is discussed later.

Any submission within M +- acceptableMAD is considered worthy of reward. Each feed provider with such a submission is awarded $R.

Any submission outside of M +- (badK * acceptableMAD) is considered worthy of penalty. Each feed provider with such a submission is penalised $P.

All other feed providers get neither rewarded or penalised. This procedure repeats every minute with a new set of submissions from each feed provider.

How are R and P calculated?

R and P are calculated using the parameters of the model in such a manner that feed providers get annual returns dependent on our requirements.

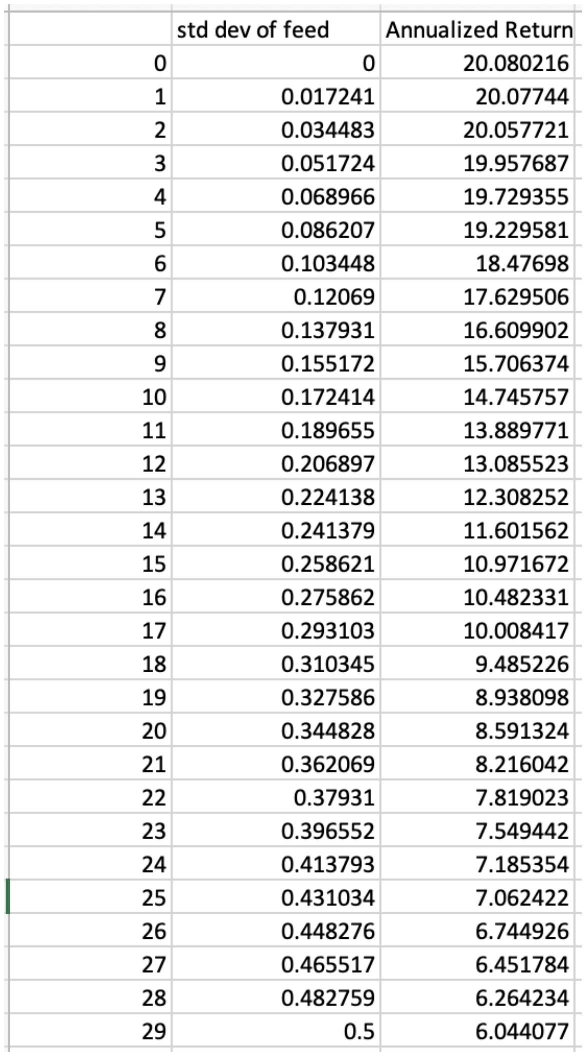
First, the deposit is calculated by dividing the TVL by nFeeds. Now, we want to set up R and P such that a perfect feed provider ends up earning around the specified goalReturns annually over the deposit, these returns get worse with declining feed quality and when the feed quality is bad enough for its standard deviation to be equal to bigLossK * acceptableMAD the feed provider loses 100% of its deposit annually. Using properties of Gaussian distributions for: a) goalReturns for perfect feed providers, and b) 100% loss for feed providers with standard deviation = bigLossK * acceptableMAD; we get two linear equations in two variables - R and P. Solving the two linear equations gives one the values of R and P for the set of parameters used.

Upcoming work includes incorporating historical submission data to decide how much to trust each feed provider, and updating the model below to include non-linear equations.

***Note:** The simulation below uses a large amount of poor-quality feed providers. As mentioned in the introduction to this section, this research is being done with the assumption of a hostile environment (i.e. constant attacks on the data feeds). As we’ve seen 50% malicious feeds can still result in accurate values, the section below aims to show punishments can still be meted out appropriately.

Simulation Scenarios

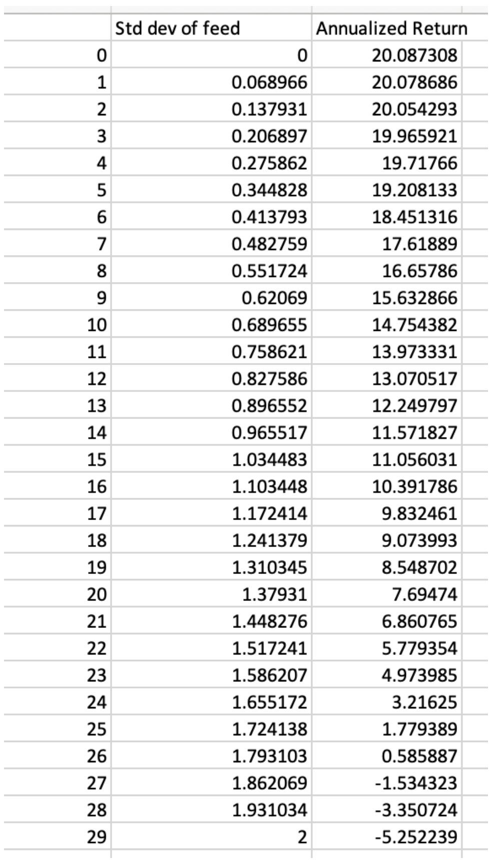
nFeeds = 30
Acceptable MAD: 2
Range of quality/standard deviation of feeds: [0 - 0.5]



	std dev of feed	Annualized Return
0	0	20.080216
1	0.017241	20.07744
2	0.034483	20.057721
3	0.051724	19.957687
4	0.068966	19.729355
5	0.086207	19.229581
6	0.103448	18.47698
7	0.12069	17.629506
8	0.137931	16.609902
9	0.155172	15.706374
10	0.172414	14.745757
11	0.189655	13.889771
12	0.206897	13.085523
13	0.224138	12.308252
14	0.241379	11.601562
15	0.258621	10.971672
16	0.275862	10.482331
17	0.293103	10.008417
18	0.310345	9.485226
19	0.327586	8.938098
20	0.344828	8.591324
21	0.362069	8.216042
22	0.37931	7.819023
23	0.396552	7.549442
24	0.413793	7.185354
25	0.431034	7.062422
26	0.448276	6.744926
27	0.465517	6.451784
28	0.482759	6.264234
29	0.5	6.044077

Here we can see that when all the feed providers are honest, feed providers that fall closest to the median get rewarded higher, while no feed providers are penalized, although those with the further from the median are not rewarded as much.

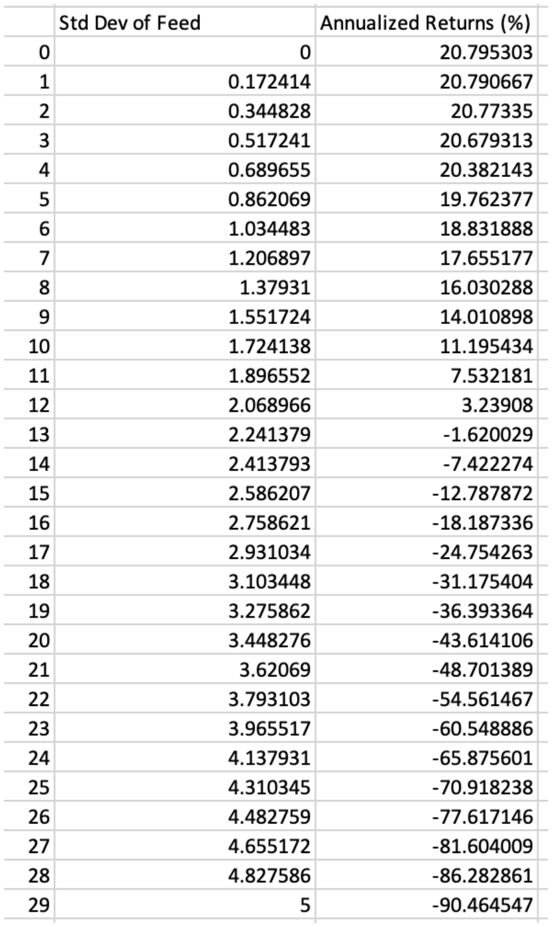
nFeeds = 30
Acceptable MAD: 2
Range of quality/standard deviation of feeds: 2



	Std dev of feed	Annualized Return
0	0	20.087308
1	0.068966	20.078686
2	0.137931	20.054293
3	0.206897	19.965921
4	0.275862	19.71766
5	0.344828	19.208133
6	0.413793	18.451316
7	0.482759	17.61889
8	0.551724	16.65786
9	0.62069	15.632866
10	0.689655	14.754382
11	0.758621	13.973331
12	0.827586	13.070517
13	0.896552	12.249797
14	0.965517	11.571827
15	1.034483	11.056031
16	1.103448	10.391786
17	1.172414	9.832461
18	1.241379	9.073993
19	1.310345	8.548702
20	1.37931	7.69474
21	1.448276	6.860765
22	1.517241	5.779354
23	1.586207	4.973985
24	1.655172	3.21625
25	1.724138	1.779389
26	1.793103	0.585887
27	1.862069	-1.534323
28	1.931034	-3.350724
29	2	-5.252239

When the range of the deviations of the submitted values is increased from 0.5 to 2, we can see those values near the edge of 2 are now being penalized, but up to 5% per year. These feeds at the tail end are still acceptable, but relatively poor quality.

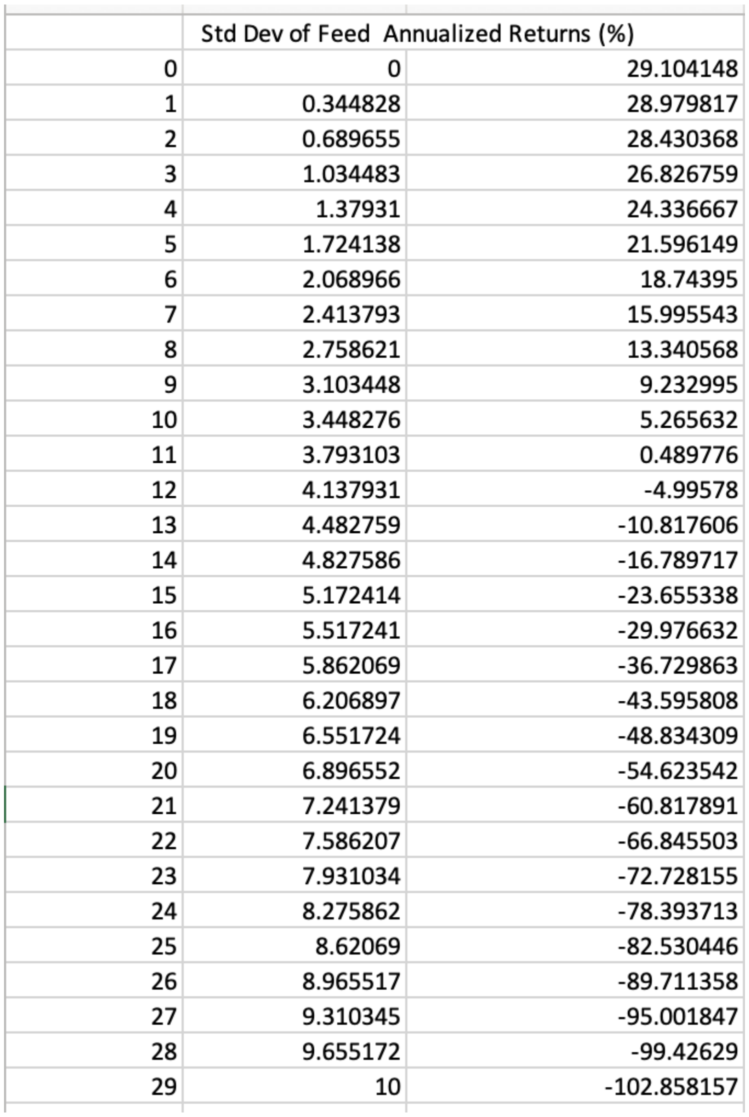
nFeeds = 30
Acceptable Deviation: 2
Range of quality/standard deviation of feeds: [0 – 5]



	Std Dev of Feed	Annualized Returns (%)
0	0	20.795303
1	0.172414	20.790667
2	0.344828	20.77335
3	0.517241	20.679313
4	0.689655	20.382143
5	0.862069	19.762377
6	1.034483	18.831888
7	1.206897	17.655177
8	1.37931	16.030288
9	1.551724	14.010898
10	1.724138	11.195434
11	1.896552	7.532181
12	2.068966	3.23908
13	2.241379	-1.620029
14	2.413793	-7.422274
15	2.586207	-12.787872
16	2.758621	-18.187336
17	2.931034	-24.754263
18	3.103448	-31.175404
19	3.275862	-36.393364
20	3.448276	-43.614106
21	3.62069	-48.701389
22	3.793103	-54.561467
23	3.965517	-60.548886
24	4.137931	-65.875601
25	4.310345	-70.918238
26	4.482759	-77.617146
27	4.655172	-81.604009
28	4.827586	-86.282861
29	5	-90.464547

Here we see a scenario in which about 40% of feeds are providing values outside the MAD. We are now beginning to see the limits of when the oracle can still function while being ‘fair’. Fair in this context means rewarding feeds with correct data and penalizing outliers. Poor quality values are not penalized as heavily as malicious data providers.

nFeeds = 30
Acceptable Deviation: 0-2
Range of quality/standard deviation of feeds: [0 – 10]



	Std Dev of Feed	Annualized Returns (%)
0	0	29.104148
1	0.344828	28.979817
2	0.689655	28.430368
3	1.034483	26.826759
4	1.37931	24.336667
5	1.724138	21.596149
6	2.068966	18.74395
7	2.413793	15.995543
8	2.758621	13.340568
9	3.103448	9.232995
10	3.448276	5.265632
11	3.793103	0.489776
12	4.137931	-4.99578
13	4.482759	-10.817606
14	4.827586	-16.789717
15	5.172414	-23.655338
16	5.517241	-29.976632
17	5.862069	-36.729863
18	6.206897	-43.595808
19	6.551724	-48.834309
20	6.896552	-54.623542
21	7.241379	-60.817891
22	7.586207	-66.845503
23	7.931034	-72.728155
24	8.275862	-78.393713
25	8.62069	-82.530446
26	8.965517	-89.711358
27	9.310345	-95.001847
28	9.655172	-99.42629
29	10	-102.858157

Once the feeds are submitting values falling way outside the acceptable MAD, we begin to see much heavier penalizations, but also rewards to very poor-quality data. This should indicate to the system that the data is unreliable.

Initial Conclusion

We the research above, we begin to get a picture of how a decentralized oracle can operate in a decentralized manner. The initial results indicate that with knowledge of a feed's acceptable MAD, the system can tell when it has been corrupted; however, even when the system is compromised with up to 20-30% of bad feeds, it can still reliably provide correct data, while penalizing bad actors and removing them from participation. Thus, we can hypothesize that not only is taking the median value of majority honest feeds likely to produce and accurate value, but that using the MAD is a good measure to understand the health of the system. It means that an attacker would need to take over enough feeds so that they all provide a value that are corrupt, but still very close to each other. With a limit of 60 feed providers per feed, once the system is in an optimal state, this would be hard to accomplish.

This research also allows the oracle to determine how much to reward honest users. When the arbitrary value of 20% profit of the deposit was chosen, it is clear that good quality feeds can be incentivized appropriately over a long time when providing good feeds. This is important because it allows the system to not provide excessive rewards, and not provide rewards that are not good enough. The profit and rewards may be tied to the value that the feed protect, and the value of the data being provided.

Attack Vectors/Downsides

The long con

It's not impossible that a single entity would choose to become an oracle to eventually defraud the system. For example, a sports feed provider might choose to reliably provide feeds all season during an NFL, and then provide incorrect values during one of the biggest betting events, the superbowl (aka the final game). One of the main defenses against this the aggregation feature. Since values such as sports scores have a MAD of 0, the attacker would have to first gain a super majority so that not only is the median 0 , but the MAD is also practically zero. Otherwise, the system would trigger the alarm by notifying smart contract, and possibly leaving it to the community to vote, and possibly slashing a large number of nodes.

Quality feeds mixed with poor quality/malicious feeds.

In a system where institutional grade feed providers are mixed with lower quality feed providers, the lower quality feed providers could reach a point where the system is corrupted, and institution grade feed providers that guarantee high quality data would be penalized. To get around this limitation, and to serve applications that cannot compromised on accuracy or speed, an additional layer of permission may be added – restriction access to feed providers of a minimum quality. This is discussed in the technical architecture section below.

Technical Architecture

To prevent a single point of failure, and to safeguard the system against malicious attackers, it's important to ensure that the system is distributed. This essentially makes Algoracle an Asynchronous Byzantine system, and as such requires some form of a consensus protocol to ensure security.

Each node should propose the median value it calculated from the data it aggregated from the feed providers, and the honest nodes of the system should agree on the value and assign that value onto the relevant smart contract. Certain feeds may not aggregate, and simply pass through the array of data, along with the source, and allow the consumer contract to handle their own aggregation.

There are three main types of contracts for each feed.

Feed contract

This contract holds either the raw array values of the submissions, or the aggregated value of the submissions (depending on the type of feed).

Reputation contract

This contract holds the acceptable median deviation values for the feed, as well as the list of nodes authorized to submit values or run nodes.

Payment contract

This contract facilitates the payment and payouts of rewards, penalties, deposits, withdrawals, etc.

Indexing Contract

This contract is simply a map of addresses that points smart contracts to the relevant feeds that consumers are interested in.

Decentralized Nodes

Decentralized nodes oversee calling the API, then placing the data onto the smart contract. Essentially, they are the interface between FPs and on chain contracts. Each node will run code that will:

A – Join the node cluster
B - Call all the endpoints on the system, and possibly aggregate the results
C – Run a VRF function to determine if it is the node selected to update the smart contract
D – Broadcast the value and the result of whether it was selected to update the contract
E – Nodes are then selected to vote to confirm the results (a subset of nodes are selected)
D – The selected node(s) can then update the Smart contract with the agreed-on value

The above methodology was inspired by the way Algorand chooses who commits blocks, and in fact uses the same open-source library that Algorand uses to select the 'lottery winner' that will propose the value, the voters who will vote on the proposal, and the certifiers before the values are written to the smart contract.

Joining the node cluster

Algoracle plans to use a tool such as serf for cluster membership and failure detection; a tool that is decentralized, fault-tolerant, and highly available. Serf is based on a gossip protocol, like Algorand and other blockchains. Nodes are required to pay the deposit, and then use their private keys to sign messages. Node members obtain a list of verified members from the reputation smart contract.

Membership

The node cluster members will always be known to each other, and every time the membership changes, a custom handles script is executed. This will help keep track of the number of nodes. Membership will also include failure detection, as nodes must exit the cluster gracefully (by notifying of withdrawal), so the cluster does not reconnect to the node.

Custom Event Propagation

Events (aka messages) are broadcast to the cluster whenever a new round begins. A new round begins after a predetermined amount of time. When accepting messages from other nodes, there are two distinct checks that are done by each node:

1 – Stateless Check: a set of checks on the message to ensure its valid. (e.g., valid hash of aggregated values, structure of message, duplicated messages, values aggregated are within the MAD etc.).
2 – Sortition Proof: a check to confirm that the sender is part of the list of verified nodes allowed to gossip on the network.

These two checks are to ensure that nodes have paid the deposit and are propagating valid data.

Aggregating the results

Valid nodes will call API endpoints using their signature. API providers documentation will show the providers how to validate the signature. This ensures the safety of the provider's API data source, by ensuring only validated nodes can make calls to it. Next, nodes can either aggregate the data, or add the raw data into an array for consumers to aggregate themselves (or select from a specific source) depending on the feed type.

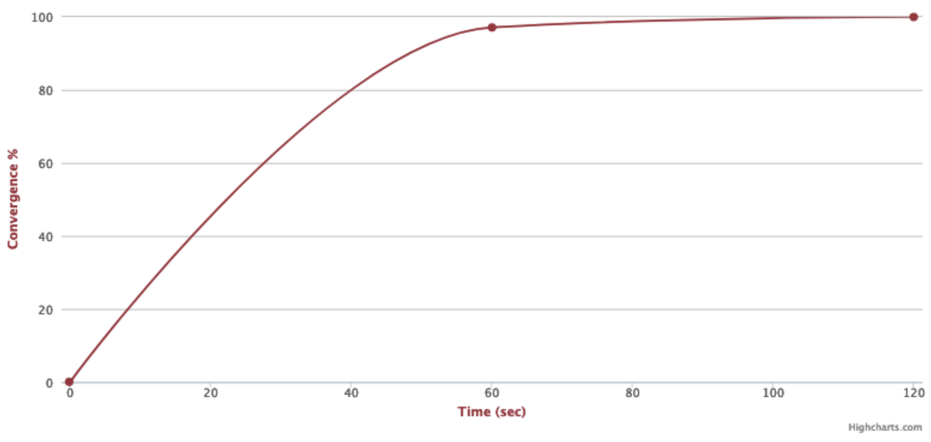
Selection Proof

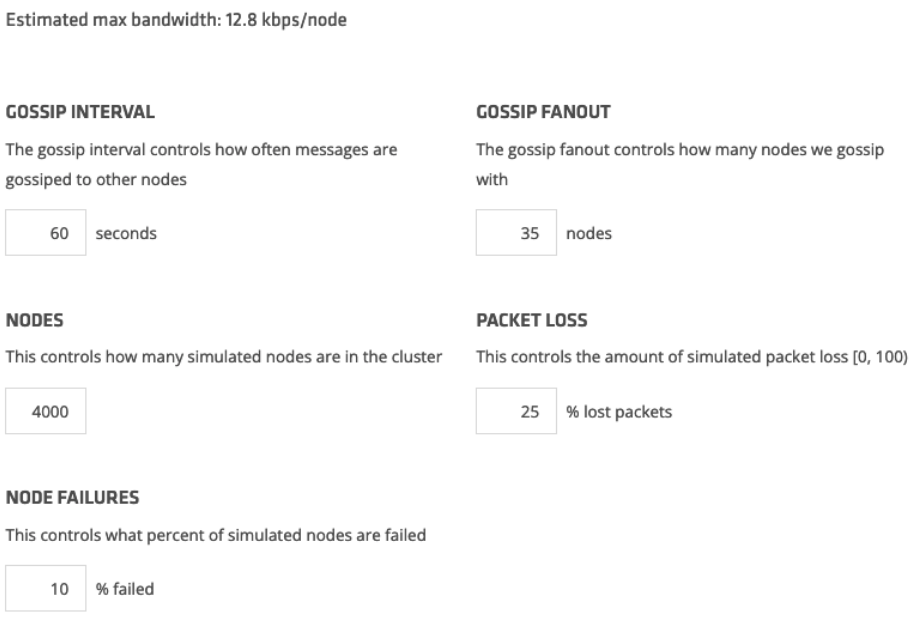
Like Algorand, Algoracle nodes run a cryptographic sortition algorithm to determine the nodes that will a) propose the value(s) to commit to the smart contract, b) vote that the value is legitimate and c) certify the values for addition onto the smart contract.

The sortition algorithm used is the open-source library that Algorand uses[10]. Each node generates a pseudo-random number with their secret key and generates a proof in addition to the random number using the previous value of the median for that feed as the seed and propagates that value to other nodes. Each node then runs a verification algorithm with the random number, public key, proof, and seed to confirm who was selected as proposers, voters, and certifiers.

Nodes will communicate with 30-40 of their peers, and only propagate valid messages. For a price feed that gossips every 30 seconds to 4000 nodes, nodes should reach a majority agreement (66.6%) and write to the smart contract within 17 seconds, even if 10% of nodes are down and a 25% packet loss. 97% agreement would be reached in under 30 seconds (see figure below).

[10] Libsodium - https://github.com/jedisct1/libsodium




Estimated max bandwidth: 12.8 kbps/node

GOSSIP INTERVAL

The gossip interval controls how often messages are gossiped to other nodes

60 seconds

GOSSIP FANOUT

The gossip fanout controls how many nodes we gossip with

35 nodes

NODES

This controls how many simulated nodes are in the cluster

4000

PACKET LOSS

This controls the amount of simulated packet loss [0, 100)

25 % lost packets

NODE FAILURES

This controls what percent of simulated nodes are failed

10 % failed

Updating the value

Once the nodes reach agreement, the value proposer(s) would then be given the rights to update the smart contract global state with the agreed-on value. The smart contracts method of Algorand allows for complex functionality by combining the functionalities of Rekeying, Logic signatures and multi-signatures.

The process above repeats indefinitely.

Attack Vectors

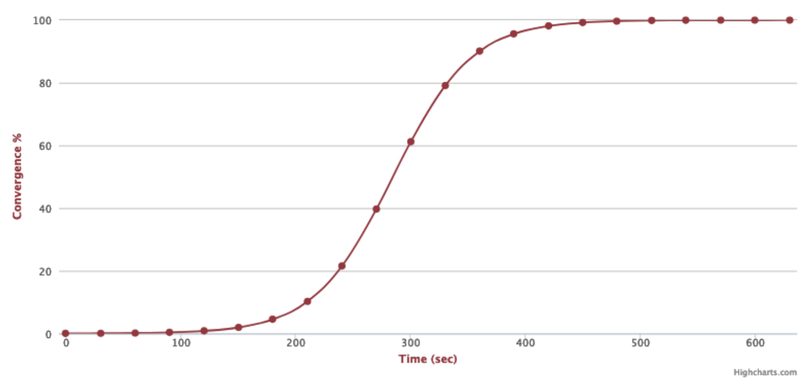
Free loading

It’s possible that a node pays the deposit, joins the network, and instead of pulling and aggregating from APIs, decides to simply act as a relay node by copying the messages sent to it. This would have the effect of at worse, exacerbating bad feeds in a low-quality environment, and at best, lowering the MAD of the feeds to slight favor the value replicated (in a situation where a subset of APIs are called by the feed providers). Although calling the APIs and aggregating the data is computationally trivial task, and there’s little to gain by free loading, it should be planned for as attackers may get creative in how the values are free loaded. As such, one method is rather than transmitting the raw values, to transmit hashes with cryptographic signatures that would be verified by other nodes. This would allow the data to remain only known to valid nodes who are authorized to call the APIs with their API keys, until the value is written to the smart contract.

Ddos

A distributed denial of service occurs when malicious or unauthorized nodes connect to a network and send unauthorized messages in order to slow down the system. Because each node’s message would not be passed on, it would take a significantly large number of nodes to attack the system. Furthermore, the size of the messages should be limited to 1-2 kb, and messages larger than that should automatically be ignored. Nevertheless, it is a real possibility of such an attack carrying out in order to delay a feed such as price feed from carrying out their duties. This is a common attack vector that even the Algorand blockchain is susceptible to.

In such a situation, a successful attack may bring down 90% of the network, and result in the remaining 10% of nodes dropping 50% packets. This would lengthen then time for the majority agreement from 17 secs to about 300 secs (about 5 minutes).



Estimated max bandwidth: 25.5 kbps/node

GOSSIP INTERVAL

The gossip interval controls how often messages are gossiped to other nodes

30 seconds

GOSSIP FANOUT

The gossip fanout controls how many nodes we gossip with

35 nodes

NODES

This controls how many simulated nodes are in the cluster

4000

PACKET LOSS

This controls the amount of simulated packet loss [0, 100)

50 % lost packets

NODE FAILURES

This controls what percent of simulated nodes are failed

90 % failed

For a worst-case scenario, this would require significant resources, while delaying feeds that report in 30 seconds to around 5 minutes. An attacker might find this useful if they have some knowledge about an assets performance in that 5 minute time period, but ultimately, once they system figures out who malicious nodes are, the attacker would then have to try again with a new set of resources. Ultimately, as any blockchain can attest to, a decentralized system is very hard to secure 100%, and the best most can do is require significant resources to corrupt for a smaller reward to be gained.

Smart contract hijacking

Because access to sign and update contracts would be granted the certifiers and proposers, a malicious node selected would need to both be randomly selected for multiple roles, which is improbable with a high number of nodes. Also, because the payment and rewards smart contract is a separate and tamperproof contract, when the next round of nodes attempt to write to the contract are locked out, this would trigger an immediate forfeit of all the staked tokens, and since the attacker would need to be controlling 5-8 malicious nodes, the penalty is multiplied. For a node system of several thousand nodes, it would be very costly to hold a large enough nodes that all the attacker's nodes are selected in the proposal, voting and certification block, and even larger still to prevent those nodes from getting slashed. And in the event of such corruption, the devaluation of the Algoracle token used to stake for the large number of nodes they have may be far greater than the value the stand to profit.

Bounty Program

In addition to the security considerations above. Algoracle also proposes a 'Bounty Program' as an additional layer of security, as well as an element of gamification and source of rewards for the community.

We except the bounty program to incentivize non stakeholders, as this adds an element of gamification. Members of the crypto community generally participate in staking and reward games, and this offers non-technical users a chance to earn rewards generally only available to technical people who can run nodes or individuals who can stake large amounts of tokens. It may also be possible for technical users being able to build bots to find bad feeds.

Note, we believe the Bounty program to be the 'last mile' of security, one where most feeds are already honest, and to be used as a deterrent against new feeds who join to slowly corrupt the network by modifying the MAD over a period of time without the system noticing.

Updates That will be added in v0.3

- Algoracle Use Cases
- System Lifecycle Diagrams
- Oracle solutions landscape
- Team Information
- Improved Design
- Asymetric API encryption details