Pythia - a decentralized market for onchain datafeeds

Justin Goro

April 2018

Abstract

The Ethereum blockchain is in need of a source of trustless, censorship-resistant oracles in order to bring to market many of the promises offered by the advent of smart contracts. Until now the design of smart contracts is such that they rely on call backs, triggered by oracles. Ideally, a smart contract should be able to dip into a stream of constantly updated data in order to execute based on external conditions. For instance, an insurance contract that can release funds when a change in financial markets occurs without direct human oversight. We offer Pythia as solution designed to elicit reliable, regular data from a source of market disciplined oracles, regulated by the Ethereum implementation of the Ulex dispute resolution mechanism. One of the innovations of Pythia is to separate the source of data from the stream, allowing the system to avoid building up a fragile reliance on any particular APIs. By requiring both oracles and consumers of data to participate in multi-step auctions for feeds, the most popular feeds will bubble up to become the most reliable, establishing an institution of objective truth that smart contract designers can treat as de facto programming primitives.

Introduction

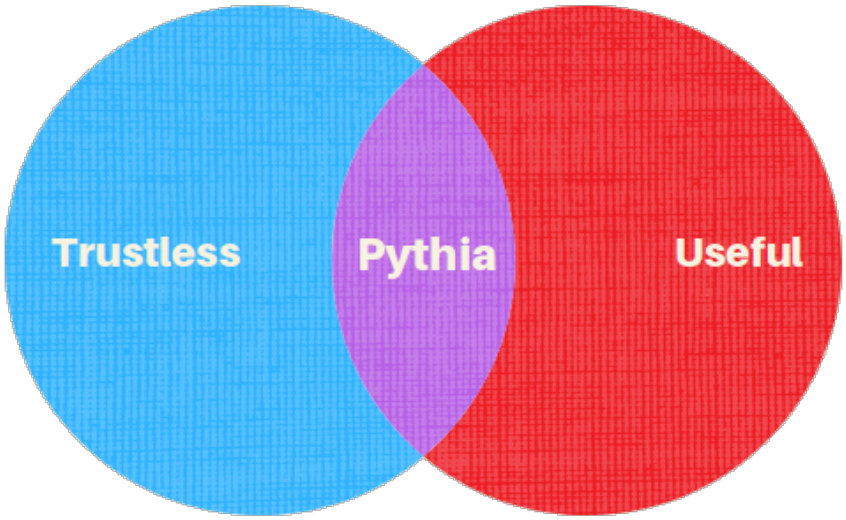
Smart contracts executing on the Ethereum blockchain have no access to events external to the network. As such, they rely on external actors (henceforth oracles) to supply external information. Since smart contracts are trustless and censorship-resistant by default, introducing a reliance on oracles negates this strength of design, precluding certain classes of use cases. Smart contract developers are currently required to choose between designing robust, trustless but ignorant contracts *and* aware contracts that are vulnerable to both trust based and censorship attacks through their reliance on oracles. Since the reliance on external, often human, actors is necessary for a smart contract to access external events, we propose that a market be created which disciplines and

decentralizes the supply of oracle data in a manner that protects data from censorship. Two levels of indirection are required to achieve not only censorship resistance but reliability of data integrity. The first is a robust market place of oracles who are required to stake deposits before selling information. The market is regulated by a decentralized, blockchain based judicial system. The second is to establish blockchain feeds which are independent and agnostic of particular sources. For instance, when establishing a feed for the latest Eth/USD price, the feed specifies accuracy and frequency but does not require oracles to reveal their sources. So long as the oracles provide reliable and accurate data, the dispute layer will act to objectively verify the integrity of the data when necessary, negating the need to rely on well known APIs.

The Name

The name Pythia (pronounced pie-thee-uh) refers to a period of divination in Ancient Greece (Forrest (1957)). Pythia was a title given to an Oracle of the god Apollo. The Pythia was a *replaceable* figure who would change from generation to generation, selected from a sample of priestesses. As such, though a particular Pythia might leave or die, the institution of Pythia was robust and lasted many centuries. The name itself refers to the monstrous python slain by Apollo and evokes imagery of a multi-headed beast, with no central point of failure. While individual Pythia weren't as revered as the Oracle of Delphi, the institution of Pythia was so respected and entrenched that many Greek scholars neglected to explain the term when referencing one. In a similar way, the platform Pythia does not rely on the reputation of esteemed APIs but instead establishes a chain of replaceable oracles who are disciplined by the mechanics of the ecosystem to provide a trustworthy institution of blockchain datafeeds. APIs will come and go but Pythia will endure.

The Oracle Dilemma



Ethereum and Bitcoin both require the consensus of nodes in order to establish a verifiable source of truth in the form of a blockchain (Nakamoto (2008)). As such, it would be impossible for nodes to include data from a source external to the network and still expect every node to verify the correctness of the data in the time it takes to publish a block. Reaching consensus on an API call requires certain layers of trust and introduces a non deterministic element to the construction of each block. For this reason, smart contracts have no native access to external events. Traditional legal contracts between parties reference external events as a matter of course. Yet, until now, trustless smart contracts have acted on nothing but events internal to the blockchain. This has narrowed the scope of possible smart contract design to a class of internally verifiable use cases such as tradeable tokens. While blockchains have been touted to replace traditional financial and insurance mechanisms, their insulation from the external world hinders the realization of this vision. For instance, a simple insurance contract designed to pay farmers in the event of a flood would require a reliable source of weather. While the funds themselves could be secured by multisignature smart contract design, the source of weather would need to be agreed upon, introducing a source of trust and 3rd party intermediation.

Stable Coins

The volatility of blockchain tokens has necessitated the need for stable coins, tokens which do not fluctuate wildly in value when compared to traditional fiat currencies such as the US dollar. The first generations of solutions offered have been to introduce tokens backed by offchain, real world assets (Anthony

C. Eufemio (2016)). These reserve tokens are vulnerable to censorship attacks through confiscation of real world assets. The MakerDAO collateralizes all of its assets on chain but the source of price feed is still maintained through a list of trusted oracles. The purpose of Pythia in the stable coin market will be to replace trusted nodes and oracles with reliable institutions of data flow, independent from the reputation of individual oracles.

Once accurate feed sources are established, smart contracts can be designed which reference a feed by its ID, a value that is invariant over time, unlike a list of ever changing oracles. In time, certain feeds will become so established that they will be treated as Ethereum primitives in contract design.

Passive and Active Oracles

The most common type of oracle to date is the active oracle, primarily because of its simplicity of design. Here, a smart contract designer exposes a public function which an oracle can trigger at will. For instance, an oracle which releases funds when triggered by a particular user will have a *release()* function exposed. The oracle is given authority to call a function on a contract and must do so at the appropriate time.

The second and more indirect type of oracle is the passive oracle. Here, an oracle continuously updates a feed of data on the blockchain which is independent of any contracts that rely on that information. A smart contract can then dip into this feed when needed without alerting the oracle. For instance, suppose an oracle provides an ongoing, minute by minute feed of the BTC price in USD into a feed contract. A consumer smart contract is then designed to release its funds when the the price falls below $1000. It has a *release()* function. Whenever a user calls the release function, the smart contract immediately requests the latest price from the BTC price feed contract and acts on it synchronously in the same block.

In the case of the active oracle, the workload of the oracle scales linearly with the number of smart contracts deployed which rely on it. It has to trigger functions on every contract when an event occurs. The passive oracle need only supply a regular feed to the blockchain, remaining ignorant of the number of dependents on its feed. The number of smart contracts relying on that feed can scale without limit. Pythia establishes a market place for strictly passive oracles.

A word on continuous vs point feeds

Aside from continuous feeds it might be desirable to establish a window of time in which one piece of information is required. For example, rather than require an ongoing exchange rate, require instead the exchange at around 12pm tomorrow. This is also consistent with a passive oracle even though the nature of the data

resembles an active oracle. The important distinction here is that the reliance injection of data can be separated from the request for that data and as such the relationship between the caller and the oracle can be decoupled.

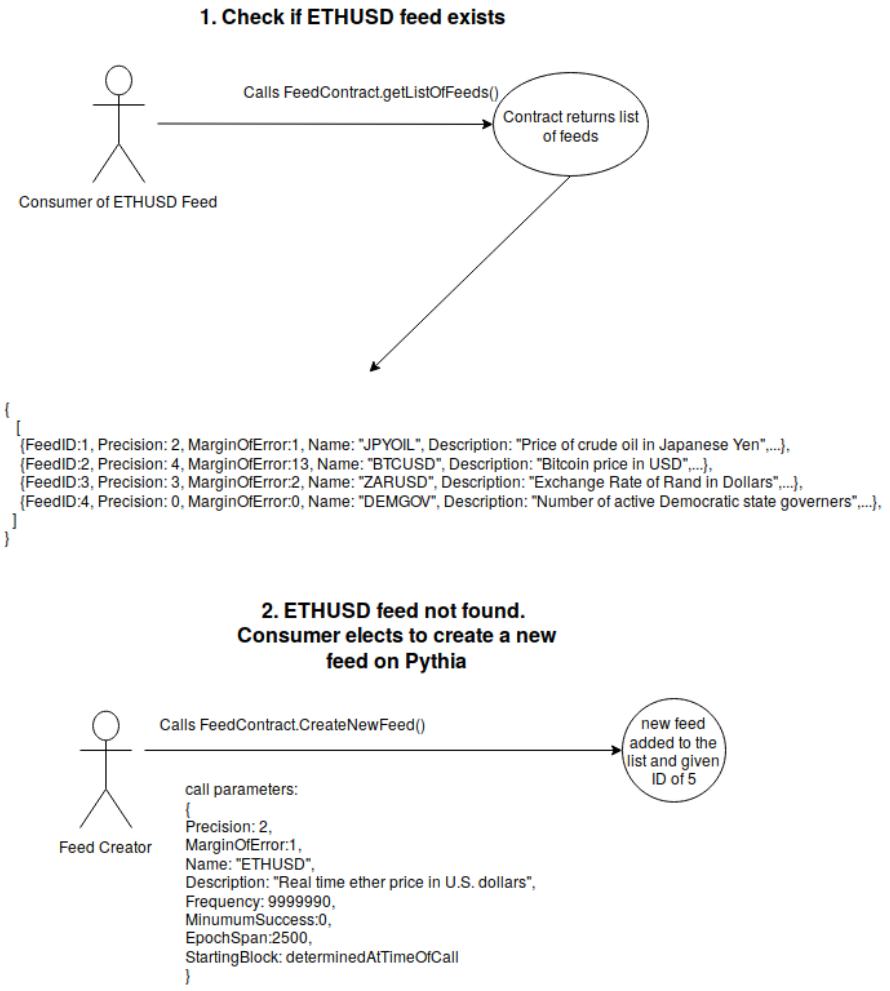


Figure 1: Registering a new feed

Contract Design

When designing a smart contract to rely on a passive oracle, the designer has to establish first that there is a feed which fits their need and secondly that the feed

is being updated with a desired frequency. Once these 2 traits are established, the contract can be designed without future upgrading necessary. This contrasts with reliance on active oracles which require constant vigilance from consumers. In the case of Pythia, a feed type such as "USD/BTC price" is given a unique ID which contract designers can use to reference its data. The ID will never change, allowing the designer to hard code the feed into their design. Pythia allows smart contract developers to outsource the establishment of reliable oracle networks, saving new projects from first curating an active community.

Solidy example:

```
address pythiaDataFeedContract;
mapping (address => uint) etherBalances;

//USD-BTC feed has an ID of 12 in the Pythia Feed contract
function ReleaseFunds () public {
    if(PythiaData(pythiaDataFeedContract).getLatestFeed(12) < 1000)
        msg.sender.transfer(etherBalances[msg.sender]);
}
```

Game Theory and Source Quality

Once desired feeds are established, would-be oracles can peruse the Feed contract for an exhaustive list of every feed. When querying the Feed contract, an oracle might get back a list such as the one below:

```
[
    {
        FeedID: 1,
        Precision: 2,
        MarginOfError:1,
        Name: "ETHUSD",
        Description: "Real time ether price in U.S. dollars",
        Frequency: 9999990,
        MinumumSuccess:0,
        EpochSpan:2500,
        StartingBlock: 124567

    },
     {
        FeedID: 3,
        Precision: 0,
        MarginOfError:0,
        Name: "REPS",
```

```
        Description: "number of republicans elected to the House of Representatives",
        Frequency: 10,
        MinumumSuccess:100,
        EpochSpan:2500,
        StartingBlock: 124567
    },
     {
        FeedID: 11,
        Precision: 2,
        MarginOfError:10,
        Name: "BTCUKP",
        Description: "Bitcoin core price in U.K. pounds",
        Frequency: 999990,
        MinumumSuccess:12,
        EpochSpan:2500,
        StartingBlock: 124567
    }
     {
        FeedID: 15,
        Precision: 2,
        MarginOfError:0,
        Name: "CLOUD",
        Description: "Is it cloudy at the moment in Brisbane (1 or 0)",
        Frequency: 0,
        MinumumSuccess:12,
        EpochSpan:10,
        StartingBlock: 124700
    }
]
```

Explanation of Feed List

First it should be noted that each feed has a unique ID, established at the time of registration. The **Precision** property refers to the number of decimal places. Since the EVM does not support floating point variables at the time of writing, all oracle data is fed without decimal places. Clients must perform the necessary adjustment based on the precision value. E.G. If a feed specifies precision of 2 then data fetched by clients must be divided by 100. **MarginOfError** is invoked only in the case of a dispute. When adjudicating on the accuracy of an oracle's feed, the judiciary will first establish an agreed upon source of truth. Following this, if the oracle strayed by less than the margin of error, the feed is considered valid. The participation of oracles in a feed will increase inversely with the margin of error since the risk of penalty is lower. However, a low **MarginOfError** will call into question the quality of the feed. Feed designers must balance these 2 variables to optimize for feed quality and regularity. The **Frequency** is the

number of predictions required per 10 million blocks (approximately 4 years). This number is used by the judiciary to determine if an oracle participated with the expected average frequency. Finally, the **MinimumSuccess** property establishes how many feeds an oracle is required to have successfully contributed to before they may contribute to the current feed. In the early days of Pythia, we expect this value to be zero but as oracles establish their track records, certain feeds can be designed to curate only the most reliable oracles. A cheating oracle can have its reputation slashed to zero by the judiciary. The last 2 properties will be explained in the subsequent section on epochs.

A corollary of the above design is that popular feeds such as the USD-ETH price can be designed with strict parameters, improving the quality of the information provided. As such, beacon feeds will emerge in the ecosystem which can be treated as de facto sources of truth for smart contracts. Less popular and more obscure feeds will attract fewer oracles. The reliability of such feeds will be questionable. If the use cases for a particular unpopular feed with lax parameters increase over time, consumers might be tempted to establish a replacement feed with tighter parameters. As such a spectrum of feeds for the same information might emerge in time, offering contract designers a choice between cheap and unreliable feeds or expensive but trustworthy sources.

Epochs

Each feed can only be supplied by one oracle at a time. The current oracle has a tenure that lasts for the duration of blocks as specified by **EpochSpan**. The first epoch starts at the property **StartingBlock**. To secure the right to supply for the duration of an epoch, oracles bid on particular epochs which are numbered in sequence. For instance, if EpochSpan is 1000 and the starting block is 0 then epoch1 is from block 0 to block 999, epoch2 is from block 1000 to block 1999 etc. Oracles will search for empty epochs and bid in an auction for a particular epoch. As epochs are filled, a queue of oracles emerges. For popular feeds, the queue of next oracles will be long enough to guarantee a sufficiently long list of filled epochs.

Point Feeds

Feed with ID 15 is different from the others in that it has a frequency parameter of zero and a very short epoch span. This is for feeds that rather than continous are once off events. In this example, the user of the feed is interested if the weather in Brisbane will be cloudy at a very particular point in time that spans 10 blocks. Since the frequency is zero, the oracle is only required to provide one data point in the form of a 1 or 0. While it may at first appear that a frequency of zero implies no data points, one of the principle rules of Pythia is that the oracle has to supply at least 1 data point. Ulex will ensure that this rule

overrides the fequency parameter of 0. The MarginOfError is also deliberately set to 0 since the answers required are either 1 or 0 with no grey area.

Point feeds will be useful in stablecoin scenarios where a holder of a coin wishes to trade in their coin from the backing cryptocurrency (such as ether) at a very particular point in time. They're only interested in the exchange rate at the time of redemption and are not concerned with receiving an ongoing stream of exchange rate data.

Source Ignorance

A smart contract relying on a feed will not be able to forecast the list of oracles. Participating in Pythia means relinquishing the notion of insisting that data be provided from a particular trusted source. Instead, designers should only trust the incentives and mechanics of the system to regulate the reliability of the data provided.

Bounty Auctions

Since oracles need to be compensated for the time and gas they spend supplying data, consumers of data can offer bounties per epoch (denominated in ether or an ERC20 token). If an oracle supplies data for the duration of the epoch and is not contested in the judicial overview system then they may withdraw the bounty reward. If more than 1 consumer offers a bounty, the value is simply added to the jackpot.

Feed Bounties and Consumer Economics

Certain feeds may be so important that communities of consumers arise who essentially crowdfund bounties, ensuring the continued integrity of a feed. In other cases, one consumer may have such a strong need to ensure integrity that all bounties are supplied by them. Other consumers may then simply free ride off of the integrity established by the sole donor consumer. The value and supply of bounties of a feed will therefore be directly proportional to its popularity. The use of the passive oracle model means that one feed can supply 1000s of smart contracts. A feed such as ETHUSD will be so well funded that many smart contract designers will be able to treat it as a permanent source of free information. The average cost of a feed therefore decreases with the number of consumers. Contrast this with an active oracle system where each reliant smart contract has to provide incentive to be activated. The average cost is constant in this case. The blockchain bloat in the active oracle model is a function of the

number of consumers but is invariant with use in Pythia. For popular feeds, the effect of Pythia will be to drive down the cost of consuming data and will reduce the redundancy of repeated data on the blockchain.

Deposit Auctions and Oracle Economics

Oracles compete to win the right to a bounty on a given epoch in return for supplying reliable data for the duration of the epoch. In order to secure the right to an epoch, Oracles bid in an auction before the epoch begins. The highest bid is locked until it is displaced by a higher bid. The oracle who placed the displaced bid can now withdraw their bid. A certain number of blocks before the epoch begins, the auction is concluded. The highest bid is now staked as a deposit for the duration of the epoch and for some time after, referred to as the epoch cool down period. At any time during the epoch or the epoch cool down, anyone can dispute the results offered by the oracle, triggering the dispute resolution mechanism. The bounty reward and deposit are then locked until the dispute has been adjudicated over. The deposit of the loser is divided amongst the judges and the other party. If the defendant oracle loses, the bounty is returned to the bounty issuer, otherwise it goes to the defendant oracle.

Dispute Resolution

The legal system follows the "loser pays" model. If an oracle is suspected of providing bad data, anyone can dispute the data. In order to do so the plaintiff has to stake a deposit equal in size to the deposit staked by the defendant oracle. This means that the cost of false accusation is the same as the cost of being a false oracle. For more popular feeds, both these costs are higher, penalizing bad behaviour more severely. While it is possible to use any dispute resolution system which embodies the user pays philisophy, Pythia is best coupled with the Ulex open source legal system (https://github.com/proftomwbell/Ulex) since it not only efficiently handles disputes of the type Pythia will raise but accomodates natural language rules such as "Oracles are required to provide at least one data point in an epoch" which judges can interpret accordingly. The rules of Pythia dispute resolution are defined according to the objective standards laid out in the feed (MarginOfError, RequiredFrequency and EpochSpan) as well as the rules specified under the Pythia community. The court will be free to decide on its source of truth. The design of Pythia will in the future allow users to bring their own legal system and attach the legal system to a particular feed so long as it complies with certain requirements for legal systems. It is likely that all legal systems used in the early stages of Pythia will be forks of Ulex. The Pythia platform will provide a set of smart contract interfaces that can be used to integrate with a custom legal system. While each feed can specify the legal

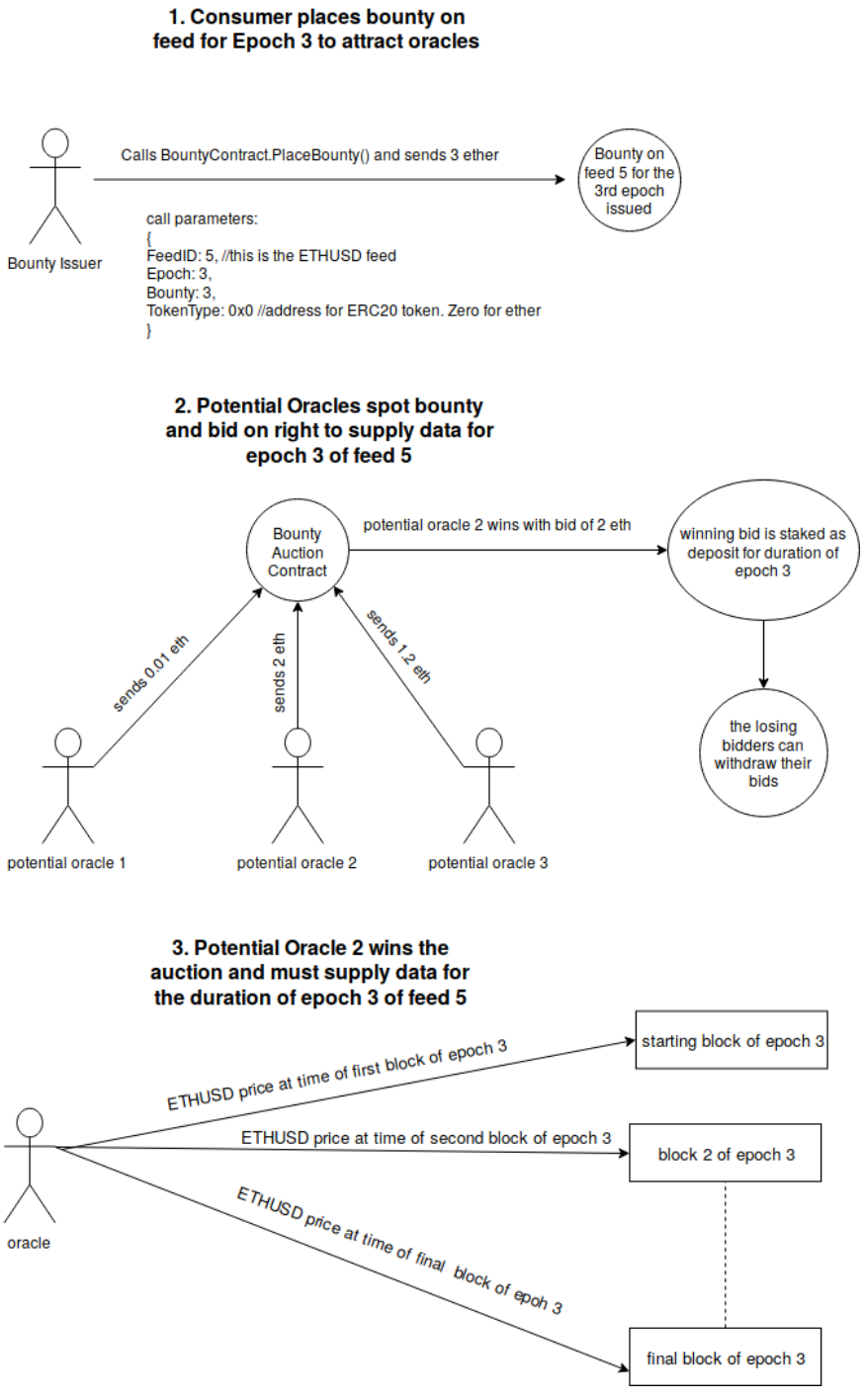


Figure 2: Attracting oracles with a bounty

system of choice, Pythia itself will be governed under Ulex and so will act as a supreme court in uncertain cases. Neglecting to specify a legal system resorts to defaulting to the version of Ulex employed by Pythia.

Suppose that a user wishes a feed to be governed by the 3 judge mechanism provided by Ulex. They would implement the interfaces correctly, deploy the necessary smart contracts and then register the entry point contract with the feed. A feed will thus be defined as

```
{
        FeedID: 110,
        Precision: 2,
        MarginOfError:1,
        Name: "TOMH",
        Description: "Number of hairstyles Tom W. Bell has had since turning 20",
        Frequency: 99990,
        MinumumSuccess:0,
        EpochSpan:2500,
        StartingBlock: 1245678,
        LegalContract: 0xF643724f52BC1316109D343E79b4Ba0dc2Faca88
}
```

The **LegalContract** property refers to the address of the deployed contract that acts as an entry point into the Ulex legal system.

Attack Vectors

The attack surface of Pythia is dependent on the vigilance of its users. The non-exhaustive list of attack vectors are:

1. Drive by shooting: An oracle supplies bogus data to an unpopular feed. This risk requires feed authors to establish a known popularity for a feed before relying on the authenticity of its contributors.

2. Faking a winning streak: Each oracle in the system has a SuccessStreak property associated with their ether wallet address. Certain feeds will specify a minimum value for this property (MinimumSuccess) in order to attract the highest quality oracles. A group of colluding oracles could establish a nugatory feed and contribute to it in order to boost their SuccessStreak, allowing them to participate in coveted feeds which have high MinimumSuccess values. Protecting against this is a special case in which users can dispute the very existence of a feed. If a feed is a bogus construct designed as a front to boost SuccessStreaks without offering useful real word data, the legal system can delete the feed as well as the SuccessStreak for any oracles which have participated in it. Faking a winning streak is consequently a high risk strategy since the colluding

oracles will expend time, gas and staked deposits in order to carry out such an attack.

3. Dead epochs: a malicious user who wishes to see a feed fail may expend resources bidding on an epoch, only to provide bogus data. The attacker in this case is willing to forfeit their deposit. This is Pythia's equivalent of a 51% attack. Currently there is no obvious solution to this. Instead, since the attack is costly, it should be unsustainable over long periods of time. It is also less likely since a feed is not associated with one company but with a fact of reality such as the gold price in Japanese Yen. The more popular a feed, the more costly it will be to attack, similarly to how the highest value blockchains are the most resilient to 51% attacks.

Long Term Scaling

In order to save oracles gas, authors of less popular feeds may wish to establish their feed on a sidechain. Future additions to Pythia will be engineered to allow for feeds to exist on sidechains such as those provided by the Loom Network. For less popular feeds or niche application feeds (such as metrics for online games), it may be desirable to keep the feed out of the mainchain. Conceptually, Pythia naturally allows for offchain scaling. However, as mentioned above, a secondary benefit will be to establish universal sources of truth for popular feeds such as ETHUSD, negating the need for each new smart contract to source its own version of the truth.

Conclusion

We have shown that the Ethereum Blockchain is in need of a class of oracle that is censorship resistant, trustless and passive. Pythia is offered as a game-theoretically robust market solution which offers secondary benefits to the scalability of the Ethereum blockchain while also providing a long term source of truth which future smart contract designers can treat as a fundamental primitive of programming to the EVM. It also has a great name.

References

Anthony C. Eufemio, A. C., Eufemio. 2016. "Digix's Whitepaper: The Gold Standard in Cryptoassets." *Historia: Zeitschrift Für Alte Geschichte*, no. 1.03 (January).

Forrest, W.G. 1957. "Colonisation and the Rise of Delphi." *Historia: Zeitschrift*

Für Alte Geschichte 6 (H. 2): 160–75.

Nakamoto, S. 2008. "Bitcoin: A Peer-to-Peer Electronic Cash System," January.