Orakl Network

Bisonai Labs

Abstract

The quantity of decentralized web3 applications is growing, accompanied by an increase in their complexity. Many of these applications now require trustworthy blockchain oracles to unlock their full potential. Oracles therefore became one of the most important infrastructure blocks for any blockchain ecosystem. There are famous reliable oracles that secure billions of dollars, however they support only a limited set of blockchains, and rarely get challenged by new oracle solutions. This leads to lack of a competition between blockchain oracles, as well as to an unequitable centralized state of blockchain infrastructure that concentrates power in the hands of a few.

This paper presents Orakl Network, a highly customizable blockchain oracle for EVM-compatible chains. The goal of the Orakl Network is to foster greater competitiveness among established blockchain oracle players. It starts by providing oracle solution tailored to a Klaytn ecosystem. The Orakl Network's solution supports both Klaytn's mainnet and service chains. The initial set of oracle services supported by Orakl Network are: Verifiable Random Function, Request-Response and Data Feed.

1. Introduction

The blockchain technology has revolutionized the way we think about trust and transparency in digital transactions. The blockchain is an isolated system without access to other data sources with abundant information like the internet for example. Blockchain's consensus ensures that all accepted transactions are valid, and submitted only by rightful account owners or smart contracts. This is possible to achieve due to deterministic nature of blockchain behavior and a full observability of its state. Over the time, web3 applications grew in their complexity and their need for an additional source of data besides the blockchain state grew as well. This led to the advent of input oracles that try to solve one of the biggest challenges facing blockchain technology: integration of off-chain data into on-chain transactions. Consequently, the presence of reliable input oracle solutions has enhanced the appeal of blockchains, empowering application builders and bolstering the competitive standing compared to blockchains without such oracles.

The goal of input oracles is to bring an off-chain data into a blockchain state in a trustful and verifiable manner. Anybody can submit off-chain data to on-chain state, however, the data transition alone is not sufficient enough for developers to integrate it. Web3 applications that integrate input oracle rely on a correctness of data provided and often use it in algorithms that secure unfathomable sums of wealth. Developers who choose to build an application requiring an input oracle on a blockchain lacking a trustful and reliable oracle face the challenge of creating their own oracle specifically for their application. This decision puts them at a disadvantage compared to similar applications that don't need to build their own input oracle. Additionally, it makes them more susceptible to blockchain oracle attacks, which in turn reduces the appeal of their application to potential users.

In this paper we introduce Orakl Network, a highly customizable blockchain oracle for EVM-compatible chains. Our goal is to bring more competition among established blockchain oracle players, and help where it is needed the most. We decided to support Klaytn blockchain at first. Klaytn with its 1-second finality, cheap gas fees and vibrant ecosystem is a good fit for our Orakl Network. We agreed to implement Orakl Network to accept service fee payments only in $KLAY and burn 50 % of them. Orakl Network supports both Klaytn mainnet, as well as its existing and future service chains. The initial set of oracle services supported by Orakl Network are: Verifiable Random Function, Request-Response and Data Feed.

This paper is structured in the following way: Section 2 discuss advantages and disadvantages of various input oracle solutions relevant to the design decision of Orakl Network for Klaytn. Section 3 explains frequently used terms in this paper. In the section 4, we describe the main components and features of the Orakl Network and how they are used to deliver trustworthy reliable data from off-chain to on-chain. Lastly, we conclude with the section 5 where we describe the next plans for the Orakl Network.

2. Background

The Orakl Network for Klaytn has unique requirements that differentiate itself from other mainstream oracles as described in the introduction section. We review relevant

blockchain oracles and explain their potential and disadvantages that led us to building Orakl Network. We emphasize that overall design decisions are highly relevant to Klaytn and would most likely result in different design if we focus on any other blockchain.

2.1. Related work

This section describes oracles that are on-boarded or in the process of on-boarding on Klaytn.

One of the most famous blockchain oracles is Chainlink [2]. Chainlink a decentralized oracle network that aims to securely connect smart contracts with real-world data and off-chain resources. It utilizes a decentralized network of nodes, reputation systems, and multiple data sources to provide accurate and reliable data to smart contracts. Chainlink is deployed on multiple blockchains, and is considered a leader in price feed services. Many prominent DeFi projects, such as Synthetix, AAVE, and Trader Joe, rely on Chainlink's price feeds to access accurate and reliable data for their operations and functionalities. Chainlink is currently deployed only on Klaytn's testnet, and requires a special token $LINK to pay for Chainlink's services. One of the Orakl Network's focus is on great user experience, and we believe that use of a native token to pay for oracle services is one such example.

Witnet [3] is a permissionless blockchain oracle that aims to provide verifiable and accurate data queries in a decentralized manner. It runs its own dedicated blockchain, incentivizing witnesses for honest participation. Randomly selected, anonymous peers aggregate data from multiple sources to establish a reliable "truth." Witnet provides randomness generation, access to off-chain data and price feed services. Witnet has been deployed on Klaytn's mainnet, but the selection of price feeds on this blockchain is very limited.

Supra Oracles utilizes a DORA protocol [6] that was designed to be resilient to Byzantine failures. The protocol uses a novel approach called agreement distance, which allows nodes to agree on a single representative value even if some of the nodes are Byzantine. Supra Oracles services include randomness generation [5] and price feeds. The collection of price feeds deployed by Supra Oracles on Klaytn is larger compared to Witnet, however, the feed selection lacks relevance in Klayn ecosystem.

3. Terminology

Every blockchain oracle uses a slightly different terminology to explain its design. To mitigate misunderstanding between Orakl Network and other blockchain oracles, we define several frequently used terms from a perspective of the Orakl Network.

Consumer represents either an EOA (externally owned account) that initiates a transaction that leads to interaction with the Orakl Network, or it can denote a smart contract itself that request a service from the Orakl Network.

Coordinator is the main Orakl Network's smart contract specific to every request-response data access approach. In section 4.1, you can learn more about `VRFCoordinator` and `RequestResponseCoordinator`.

Oracle is a single entity (also called a **node operator**) that plays a role of data relayer to Orakl Network. The oracle entity can be different for every Orakl Network service, but it always must be approved before it can submit any data to chain. Oracle is implemented as an off-chain component, and in this paper we use terms "oracle" and "off-chain oracle" interchangeably.

4. Design

Orakl Network consists of on-chain and off-chain components that work together to provide off-chain data access to on-chain consumer smart contracts. At the current phase, Orakl Network supports two different data access approaches: *request-response* and *immediate read*.

The request-response data access approach is initiated by a consumer smart contract, and consequently fulfilled by the Orakl Network node operators. Consumer smart contract can either request the *Orakl Network Verifiable Random Function* service to generate provably random numbers, or *Orakl Network Request-Response* service to bring any off-chain data accessible through HTTP(S) protocol back to blockchain. Both services are charged using Klaytn's native token $KLAY.

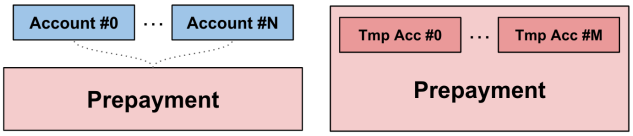
The immediate read data access approach is utilized by the *Orakl Network Data Feed* service which relays an off-chain data streams to aggregator smart contracts. The data stream is relayed by a limited set of trustworthy node operators. Each node operator independently relays data to a shared aggregator contract, creating an auditable trail that facilitates subsequent investigations in the event of malicious data submissions. All consumer smart contracts have free access to the entire pool of relayed data streams without any associated charges.

The rest of this section discusses in detail on-chain components (section 4.1), off-chain components (section 4.2) their integration for Klaytn's mainnet, and service chains (section 4.3).

4.1. On-chain components

Orakl Network smart contracts were designed from first principles and with simple goals in mind: ease of use, and high interoperability between distinct services.

We acknowledge that a very large portion of web3 developers have experienced a Chainlink oracle, and therefore are familiar with its interface. We decided to closely follow Chainlink's interface for a smooth on-boarding of developers coming from Chainlink to the Orakl Network. However,



(a) Permanent account (b) Temporary account

Figure 1. Orakl Network's payment option design

there are cases where we slightly diverge from Chainlink's interface in order to provide a better developer experience. We highlight such cases when we describe relevant smart contracts.

In the following subsections we discuss in detail smart contracts related to payments (`Prepayment`, `Account`), generation of random numbers (`VRFCoordinator`), access to off-chain data (`ReqeustResponseCoordinator`), and frequently updated data streams (`Aggregator`, `AggregatorProxy`).

4.1.1 Payments

The request-response data access approaches require users to pay for its services. Payments in the Orakl Network are handled through the `Prepayment` smart contract which ensures that consumer requesting a service has account with enough $KLAY, and transfers a service specific fee from consumer's account to oracle upon a successful fulfillment. The Orakl Network implements two types of accounts: permanent and temporary.

Permanent account (figure 1a) is designed for frequent Orakl Network users or users that want to share the account with multiple other consumer smart contracts. Consumer that creates the permanent account through the `Prepayment` smart contract becomes an account owner. Account owner can deposit/withdraw $KLAY from account, and update a set of smart contract consumers that can request Orakl Network services through the account. Every permanent account is represented by a separate `Account` smart contract.

Temporary account (figure 1b) does not require consumer to deploy a separate smart contract. The account is abstracted inside of `Prepayment` smart contract, and with every request, coordinator creates a new temporary account.

4.1.2 Verifiable Random Function

The Orakl Network Verifiable Random Function (VRF) [4] is a type of cryptographic function that utilizes a public (PK) and private key (SK) system to generate provably random numbers. Orakl Network separates the VRF computation to on-chain and off-chain part (described at section 4.2.1). Public VRF key and its hash (`keyHash`) must be registered within a `VRFCoordinator`, an on-chain implementation of VRF computation. Private key is known only to the off-chain VRF oracle.

VRF coordinator accepts requests from consumers, handles service fees through `Prepayment` contract, generates initial seed, verifies oracle submission, and concludes by fulfilling random numbers requests. During the VRF request, coordinator contract generates a pre-seed (computed as `uint256(keccak256(abi.encode(keyHash, sender, accId, nonce)))`) from consumer request parameters. The pre-seed is delivered with other relevant parameters to the off-chain oracle. Then, the off-chain oracle produces an $alpha$ seed by hashing a pre-seed with block hash number of the request, and computes a VRF hash from $alpha$ and oracle's VRF private key.

Once the off-chain computation is finished, oracle submits a proof pi and a VRF hash $beta$ to the VRF coordinator for verification. The proof pi allows coordinator holding the public key PK to verify that $beta$ is the correct VRF hash of input $alpha$ under key PK. If the verification process succeeds, VRF coordinator converts proof pi to initial randomness and use it to generate random values that are passed to consumer smart contract.

4.1.3 Request-Response

The Orakl Network Request-Response service enables consumer smart contracts to get an access to off-chain data directly from chain. Consumers can create an on-chain request and submit it to `RequestResponseCoordinator` contract, an on-chain coordinator for Request-Response service. Coordinator accepts requests from consumers, handles service fees through `Prepayment` contract, and concludes by submitting the final responses by an off-chain oracle (described at section 4.2.2).

The on-chain request metadata are serialized as Concise Binary Object Representation (CBOR) [1]. The metadata includes the following parameters: HTTP(S) address of an off-chain API server, list of postprocessing functions that are applied to the response from the API server, expected response value data type, and a number of requested responses. Each of these parameters have an impact on the final results submitted back to consumer smart contract. The API server must be accessible to the off-chain oracle while processing the request. Parsing rules must be valid and applicable to the response generated by the API server. The expected response value data type affects which function selector is used by the off-chain oracle for submission. Consumers can request to receive an aggregated value computed from multiple submissions by unique off-chain oracles. This aggregation feature is supported only for a limited set of data types under predefined conditions that are

displayed in table 1.

4.1.4 Data Feed

The Orakl Network Data Feed service distills various data sources representing the same information into a single scalar that is then submitted to blockchain. The submission frequency depends on data feed settings, and can be controlled by either time delay, or scalar deviation with respect to previous submissions. We aim to make Orakl Network Data Feeds a single source of on-chain truth for important data streams, and in order to achieve that we accept only publicly known trustworthy off-chain oracle operators. The access to Orakl Network Data Feed service is free of charge, because we believe that this strategy will help to establish a presence of Orakl Network within a larger blockchain landscape.

A single data feed is associated with `Aggregator` and `AggregatorProxy` smart contracts. They are used for aggregation of submitted scalars by individual oracles, and to provide a user-friendly interface for consumer applications, respectively. Every data feed accepts submissions only from a curated set of oracles. All submissions are assigned to a round which represents a specific timespan during which oracles are supposed to provide their latest off-chain observations. New round can be initiated by any oracle after the previous round aggregate becomes outdated.

Consumers can access the latest submission aggregates when the number of submission reaches above a predefined threshold. If a consumer needs to access historical data, all aggregates for each historical round are easily accessible on-chain as well.

4.2. Off-chain components

The design of Orakl Network off-chain components is very flexible, and can be applied across all three supported services. The main building blocks (figure 2) for each off-chain service are: *listener*, *worker*, and *reporter*.

Listener and reporter are customizable reusable components whose main purpose is to reliably catch events emitted by smart contracts of interest, and successfully submit responses back to coordinators, respectively.

Workers are different for every Orakl Network service (details can be found at following subsections 4.2.1, 4.2.2 and 4.2.3), but they are all connected to their listeners and reporters via designated job queues. This design eliminates a single point of failure of individual services while allowing for an easy scalability.

4.2.1 Verifiable Random Function

Every VRF oracle that wants to participate in Orakl Network generates a public VRF key (PK), a private VRF key

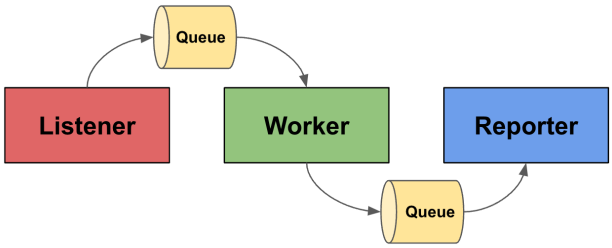


Figure 2. Orakl Network high-level off-chain architecture

(SK) at first. Then it must be registered to on-chain VRF coordinator (details at section 4.1.2). Off-chain VRF Worker receives requests through VRF Listener that captures events emitted by VRF coordinator. Every VRF request includes a hash of VRF PK that determines which oracle can fulfill the request. Oracles that do not possess corresponding VRF SK are not able to provide such a proof pi and a hash $beta$ that would pass the on-chain verification process.

The oracle, who holds VRF SK that corresponds to the requested key hash, uses it to hash the input $alpha$ (described at section 4.1.2) and produce a VRF hash output $beta$ by applying the VRF_{hash} algorithm. This process is deterministic and consistent, meaning that the same output will be produced given the same input. Additionally, the VRF oracle uses the VRF SK to construct a proof pi that the output $beta$ is the correct hash output, by applying the VRF_{prove} algorithm. The VRF_{hash} algorithm is defined in such a way that it can be deterministically obtained directly from the proof value pi by using the function $VRF_{proof\_to\_hash}$. This means that the VRF_{hash} algorithm is defined as follows:

$$VRF_{hash(SK,alpha)} = VRF_{proof\_to\_hash(VRF_{prove(SK,alpha)})}$$

After the proof pi and hash $beta$ are generated, VRF worker prepares a fulfillment transaction and submits it to job queue connected to the VRF Reporter.

4.2.2 Request-Response

The Orakl Network Request-Response can be fulfilled by any oracle registered inside of `RequestResponseCoordinator` contract. This unrestricted approach incentivizes off-chain oracles to respond to requests swiftly in order to receive reward for their service. The quality of responses is curated by allowing to participate only a limited set of trustworthy off-chain oracles.

The lifecycle of Request-Response Worker starts after it receives a request captured by the Request-Response Listener. The worker deserializes encoded request, and attempts to access the API server specified in the request metadata. If the server does not respond, the worker stores

Data Type	Aggregation	Conditions
`int256`	Median for `int256`	$numResponse \leq numOperator/2$
`uint128`	Median for `int256`	$numResponse \leq numOperator/2$
`boolean`	Majority Voting	$(numResponse \leq numOperator/2) \wedge (numResponse \bmod 2 \neq 0)$
`bytes32`		support for single response only
`bytes`		support for single response only
`string`		support for single response only

Table 1. Orakl Network Request-Response supported data types and aggregation conditions

the response error code for consumer to see, and does not continue in request processing. If the server responds successfully, the worker applies postprocessing functions to the server response. The currently supported postprocessing functions are as follows:

- `path` - list of keys for walk through input JSON
- `index` - access n-th item in the input list
- `mul` - multiply input with arbitrary number
- `div` - divide input with arbitrary number
- `pow10` - compute a power of 10 from input
- `round` - apply a round operation on input

If any of the postprocessing functions fail, the worker terminates the request processing, and stores the error for consumer to see. After all postprocessing function are successfully applied, the worker prepares a transaction payload. The function selector is determined by the expected response value data type from request metadata, and the submission value is the output of the last postprocessing function. Eventually, the worker passes the transaction payload to the Request-Response Reporter, and waits for another job from the Request-Response Listener.

4.2.3 Data Feed

The design of Data Feed Worker is more complex compared to VRF and Request-Response workers. The worker perpetually listens to various data streams for every data feed it supports, and computes their aggregates. We do not decide for oracles which data sources to use, but we prepared a recommended set for every supported data feed at `https://config.orakl.network`.

There are two main processes inside of the Data Feed Worker: *heartbeat* and *deviation*. They interact with each other indirectly through the `Aggregator` contract, and maintain a synchronization between all oracles and their on-chain submissions.

The heartbeat process is regulated by a unique heartbeat parameter for each data feed configuration. This parameter sets the maximum allowable submission delay following the completion of the last round. The oracle monitors the latest submission time and submits the most recent aggregate once the heartbeat delay has elapsed.

The deviation process is controlled by relative and absolute threshold parameters that can be configured for every data feed individually. As the oracle is aware of all deviations in data streams, it can promptly respond and submit the updated information when the data changes beyond a predefined relative threshold. Some of the data feeds have a lower bound of 0 value. An absolute threshold parameter is employed to determine whether a significant positive change occurred after the value has reached the minimum. Similar to the relative threshold, once the absolute threshold is reached, the oracle submits the most recent aggregate to the chain.

The Data Feed Worker receives information about new round openings through the Data Feed Listener, and every submissions is executed by the Data Feed Reporter.

4.3. Service chain

Service chain is Klaytn's solution to a scaling problem, which is commonly called L2 solution. It is designed for web3 developers who need high TPS, minimal transaction fees, or data privacy. Service chain solution can be used to build its own ecosystem inside of Klaytn realm, or create single-purpose web3 applications.

One of the design goals of the Orakl Network was to create a blockchain oracle from which whole Klaytn ecosystem can benefit and not only Klaytn mainnet. We designed a solution (figure 3) which allows for an easy on-boarding of new service chains and has minimal impact on Orakl Network solution for mainnet. The following sections describe additional smart contracts (section 4.3.1 and 4.3.2) and off-chain components are described at section 4.3.3.

4.3.1 Registry

The Orakl Network Registry is a singleton smart contract on Klaytn mainnet that holds information about service chains

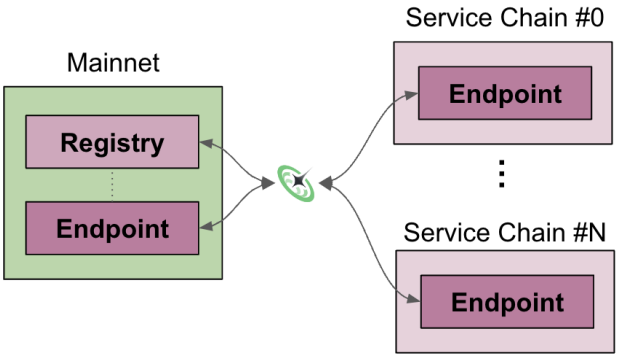


Figure 3. Orakl Network integration with service chains

and consumer accounts that want to utilize Orakl Network. To get access to Orakl Network, service chain owners must be registered to registry. The registration process is divided into two steps: `proposal`, and `approval`. The proposal includes service chain metadata that are verified and approved if everything is correct. Proposal metadata is composed of chain identifier, public JSON-RPC endpoint and address of Orakl Network Endpoint (details at following section 4.3.2) on service chain.

Orakl Network VRF and Orakl Network Request-Response are paid services and in order to use them, one must create a permanent account on mainnet and register it within the registry. The account owner can be either the service chain owner herself, or the consumer which requests an oracle service from a service chain.

4.3.2 Endpoint

Endpoint represents an interface for communication between the mainnet and a service chain. Service chain owner is responsible for deploying `Endpoint` smart contract on service chain, and for registration of endpoint address to the Orakl Network Registry on mainnet (described at section 4.3.1).

Consumers that want to use Orakl Network VRF or Orakl Network Request-Response, issue their request from service chain to the local endpoint which emits an event containing service request details. The service request details are safely and reliably transferred to mainnet's endpoint using Orakl Network Interchain Message Protocol (details at 4.3.3). Upon receiving the message at the mainnet's endpoint, request is relayed to appropriate Orakl Network service on behalf of service chain consumer. The payment for Orakl Network service is facilitated through permanent account (described at section 4.1.1) located on the mainnet. Service chain owner can either subsidize all requests that come from the service owner's chain, or consumers can create their own accounts and register them within the Orakl Network Registry. The permanent account has to be created prior to service request and must have balance high enough to pay for requested service. After the permanent account is created, and sufficient amount of $KLAY is deposited, account owner must add the address of mainnet `Endpoint` to the set of account consumers that can apply for Orakl Network services via that account. Depending on the service chain rules, mainnet's endpoint determines the appropriate account when requesting for a service. Request is processed off-chain, and oracle response is delivered to the mainnet endpoint. The mainnet endpoint creates a message event that contains oracle fulfillment details. Then, Orakl Network Interchain Message Protocol transfers the message from mainnet endpoint to the service chain endpoint that requested for a service. Finally, the service chain endpoint submits Orakl Network response to consumer smart contract where the service request originated.

4.3.3 Interchain Message Protocol

5. Conclusion

This paper introduced Orakl Network, a highly customizable blockchain oracle solution for EVM-compatible chains. We explained the importance of oracles for web3 applications, how lack of them leads to industry monopolization, and how it lowers a web3 application security when builders are forced to create and maintain their own oracles. We described in detail how the current design and implementation of the Orakl Network is tailored to Klaytn ecosystem. The paper proposed an integration between mainnet blockchain oracle and interchain message protocol to expand blockchain oracle support beyond the mainnet.

As part of our future work, we intend to strengthen our foothold in modularized on-chain and off-chain components. We believe that this will make Orakl Network more competitive and help Orakl Network to enter blockchain ecosystems with already established oracle players.

Disclaimer

The Orakl Network is under active development, and some information in this paper might be outdated. To learn about the latest updates and development, please visit `https://orakl.network`.

References

[1] Carsten Bormann and Paul E. Hoffman. Concise Binary Object Representation (CBOR). RFC 8949, Dec. 2020. 3

[2] Lorenz Breidenbach, Christian Cachin, Alex Coventry, Steve Ellis, Ari Juels, Benedict Chan, Farinaz Koushanfar, Daniel Moroz, Florian Tramer, Andrew Miller, Sergey Nazarov, Brendan Magauran, Alexandru Topliceanu, and Fan Zhang.

Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks. https://chain.link/whitepaper, 2021. 2

[3] Adán Sánchez de Pedro, Daniele Levi, and Luis Iván Cuende. Witnet: A decentralized oracle network protocol. https://witnet.io/witnet-whitepaper.pdf, 2017. 2

[4] Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Včelák. Verifiable Random Functions (VRFs). https://datatracker.ietf.org/doc/draft-irtf-cfrg-vrf/10/, 2022. Work in Progress. 3

[5] Supra Research. Supra vrf service. https://supraoracles.com/docs/SupraOracles-VRF-Service-Whitepaper.pdf. 2

[6] Supra Research. Dora: Distributed oracle agreement. https://supraoracles.com/docs/SupraOracles-DORA-Whitepaper.pdf, 2023. 2