OathProtocol Design Doc

Author: Hongwei Wang <hwwang@oaths.io>
Reviewer: Yin Xu <yxu@oaths.io>
Status: Draft
Last modified: 09/29/2018

Version	Author	Change date	Change log
0.1	hwwang@oaths.io	09/21/2018	Initial Draft
0.2	hwwang@oaths.io	09/29/2018	Adds off-chain OPP

Objective

This design document describes the semi-decentralized OATH Protocol.

A decentralized Application (dApp) can interact with OATH Protocol Contract (OPC) running in Ethereum Virtual Machine (EVM) and file disputes via OPC to OATH Protocol Platform (OPP) when necessary. Arbitration runs in OPP, and OPP callbacks the progress and result to the filing contract.

Though a fully decentralized blockchain governance and dispute resolution is possible, it is out of scope of this design document.

Background

In blockchain, smart contracts are executed within a closed environment virtual machine, which is an isolated system that has no interaction with external environment. This means that smart contracts are unable to communicate directly with external world or obtain any outside information.

Therefore, there are generally two types of blockchain governance systems:

- Fully decentralized, such as Kleros[1]. Disputes, juror participation, and decisions all happen on the blockchain.
- Semi-decentralized, such as OATH Protocol[2] v1. DApps integrate with the Oracle, while the the governance and dispute resolution process runs off-chain. The Oracle serves as a bridge between smart contract and the outside world.

Overview

An overview of the dApp smart contract's interaction with OATH Protocol Platform:

[1] Kleros · The Blockchain Dispute Resolution Layer https://kleros.io/
[2] OATH Protocol https://oaths.io/

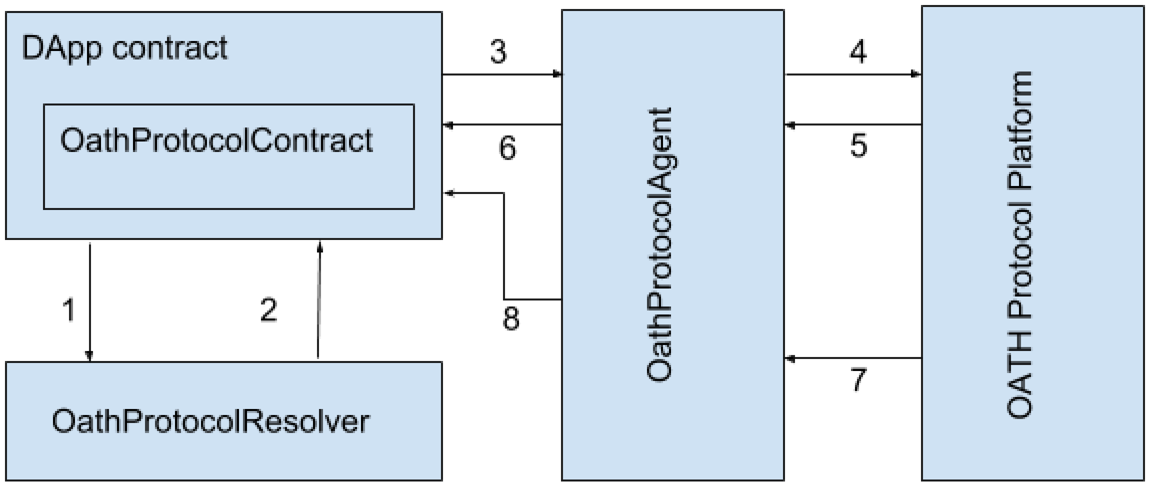

Figure. OATH Smart Contract integration

1. DApp inherits OPC and asks OathProtocolResolver(OPR) for OathProtocolAgent (OPA)
2. OPR returns the OPA instance in current network
3. DApp issue dispute creation request to OPA
4. OPA emits onDisputeFiled event, which is caught by OPP
5. OPP creates the dispute case internally and callbacks OPA about onDisputeCreated
6. OPA relays onDisputeCreated to DApp contract
7. OPP callbacks OPA with onDisputeResolved with the ruling
8. OPA relays onDisputeResolved to DApp contract

In this semi-decentralized design, OATH Protocol will provide the following components in EVM:

- OathProtocolContract (OPC), a base smart contract that Dapp can directly inherit. It provides functions like createDispute and callbacks ready to use such as onDisputeResolved.
- OathProtocolAgent (OPA), a smart contract runs in EVM that OPC will directly interact with. OATH Protocol Platform (OPP) will listen for events emitted from this contract. This smart contract forwards requests from DApp to OPP and routes callbacks from OPP to the filing contract.
- OathProtocolResolver (OPR), a smart contract runs in EVM to resolve the active OPA instance running in current Ethereum network. OPC would rely on this contract to get the actual OPA. OATH Protocol can set OPA in different network, useful to upgrade OPA without breaking any DApp inherits OPC.

When OPP receives DisputeFiled event emitted from OPA contract, it creates a dispute case internally and callbacks onDisputeCreated on OPA following by onDisputeResolved when ruling is settled by selected jurors. OPP runs off-chain.

Detailed design

On-chain integration[3]

OathProtocolResolver (OPR)

This smart contract provides the flexibility for OATH Protocol to replace the existing OathProtocolAgent without breaking any existing DApp integration, as long as the OathProtocolContract interface itself remains unchanged.

The OPR interface exposes one method that returns OathProtocolContract instance

```
interface IOathProtocolResolver {
    /**
     * @return IOathProtocolContract agent in current network.
     */
    function getOathProtocolAgent() external returns(IOathProtocolContract);
}
```

Actual OPR implementation would allow ownership transfer and etc.

```
contract OathProtocolResolver is IOathProtocolResolver {

    address public owner = msg.sender;

    address public oathProtocolAgent;

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    function setOathProtocolAgent(address _address) public onlyOwner {
        oathProtocolAgent = _address;
    }

    function transferOwnership(address newOwner) public onlyOwner {
        owner = newOwner;
    }
```

[3] https://github.com/OathsIO/OathProtocol/tree/master/ethereum-api

```
    function getOathProtocolAgent() external returns(IOathProtocolContract) {
        require(
            oathProtocolAgent != 0,
            "OATH Protocol agent is not initialized in current network"
        );
        return IOathProtocolContract(oathProtocolAgent);
    }
}
```

IOathProtocolContract (OPC interface)

This smart contract interface should be implemented by both OathProtocolAgent (OPA) and OathProtocolContract (OPC) since both contracts share the same functionalities such as createDispute and receives the same set of callbacks from OPP.

```
contract IOathProtocolContract {

    enum Ruling {
        Unknown,
        Tie,
        LeaningProsecutor,
        LeaningDefendant
    }

    /**
     * @dev Event emitted after filing a dispute.
     * OATH Protocol listens for this event filed from agent.
     */
    event DisputeFiled(
        address disputeContract,
        address prosecutor,
        address defendant
    );

    /**
     * @dev Create a new dispute in OATH Protocol.
     * This is called by integrated contract to agent and result
     * will be delivered via onDisputeCreated callback.
     */
    function createDispute(
        address prosecutor,
        address defendant
    )
        external
        payable;

    /**
     * @dev Callback after a dispute is created in OATH Protocol.
     * OATH Protocol issues this callback to agent, dispute result
     * will be delivered via onDisputeResolved callback.
     */
    function onDisputeCreated(
        string disputeId,
        address disputeContract
    )
        external;

    /**
     * @dev Callback after a dispute is resolved in OATH Protocol.
     * OATH Protocol issues this callback to agent.
     */
    function onDisputeResolved(
        string disputeId,
        address disputeContract,
        Ruling ruling
    )
        external;

}
```

OathProtocolAgent (OPA)

This smart contract implements IOathProtocolContract, serves as a transparent proxy between DApp and OPP. From the DApp's point of view, OPA is the only smart contract it interact with. While from OPP's perspective, OPA is the smart contract it listens for event and files callbacks against.

```
contract OathProtocolAgent is IOathProtocolContract { // incompleted

    address public owner = msg.sender;

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    modifier fromContract {
        require(getCodeSize(msg.sender) > 0);
        _;
    }

    function createDispute(
        address prosecutor,
        address defendant
    )
        external
        fromContract
        payable {
        emit DisputeFiled(msg.sender, prosecutor, defendant);
    }

    function onDisputeCreated(
        string disputeId,
        address disputeContract
    )
        external {
        // silence compilation warnings.
        disputeId; disputeContract;
    }

    function onDisputeResolved(
        string disputeId,
        address disputeContract,
        Ruling ruling
    )
        external {
        // silence compilation warnings.
        disputeId; disputeContract; ruling;
    }

    function getCodeSize(address addr) internal view returns(uint _size) {
        assembly {
            _size := extcodesize(addr)
        }
    }
}
```

OathProtocolContract (OPC)

This is the base smart contract that DApp should inherit and integrate with. It finds the OathProtocolResolver (OPR) in current network under the hood.

```
contract OathProtocolContract is IOathProtocolContract { // incompleted
```

```
/**
 * @dev Mapping from network id to IOathProtocolResolver
 * TODO: Deploy resolvers and have the mapping here.
 */
mapping (uint => address) internal resolvers;

IOathProtocolResolver internal oathProtocolResolver;

enum Stage {
    Unknown,
    DisputeFiled,
    DisputeCreated,
    DisputeResolved
}

Stage public stage = Stage.Unknown;

address public owner = msg.sender;

modifier onlyOwner {
    require(msg.sender == owner);
    _;
}

modifier fromOathProtocol {
    require(oathProtocolResolver != address(0));
    oathProtocolAgent = oathProtocolResolver.getOathProtocolAgent();
    require(msg.sender == address(oathProtocolAgent));
    _;
}

modifier atStage(Stage _stage) {
    require(stage == _stage);
    _;
}

modifier requireAgent() {
    if (oathProtocolResolver == address(0)) {
        oathProtocolResolver = getOathProtocolResolver();
        require(oathProtocolResolver != address(0));
        require(oathProtocolResolver.getOathProtocolAgent() != address(0));
    }
    _;
}

function createDispute(
        address prosecutor,
        address defendant
    )
        external
        atStage(Stage.Unknown)
        requireAgent
        payable {
        stage = Stage.DisputeFiled;
        oathProtocolResolver.getOathProtocolAgent().createDispute(
            prosecutor,
            defendant);
    }

    function onDisputeCreated(
        string disputeId,
        address disputeContract
    )
        external
        atStage(Stage.DisputeFiled)
        fromOathProtocol {
        stage = Stage.DisputeCreated;
        // silence compilation warnings.
        disputeId; disputeContract;
    }

    function onDisputeResolved(
        string disputeId,
        address disputeContract,
        Ruling ruling
    )
        external
        atStage(Stage.DisputeCreated)
        fromOathProtocol {
        stage = Stage.DisputeResolved;
        // silence compilation warnings.
        disputeId; disputeContract; ruling;
    }

    function getOathProtocolResolver() internal {
        // TODO: find oathProtocolResolver in current network
    }
}
```

Off-chain OATH Protocol Platform (OPP)

Points and credit level

OATH introduces a Credit Level System to encourage and enforce reasonable and fair decision making by the jurors. The system has 20 levels, and jurors with higher credit levels earn more heavily weighted token awards for services performed on the platform.

New jurors, after verification and evaluation by a number of statistical methods, are assigned an initial credit level, from 1 to 5. Jurors may level up once they reach certain point thresholds. Credit level is a Fibonacci function of points as following:

Credit level	Points	Points to level up
1	1	1
2	2	2
3	4	3
4	7	5
5	12	8

There are several ways to earn/lose points:

- Voting: Correctly deciding case outcomes (voting for the prevailing party) has the biggest impact on credit scores. Voting for the winning party earns 2 points, whereas voting for the losing party subtracts 2 points. To maintain the fairness and stability, platform will greatly decrease the credit scores of jurors who make continuous wrong choices.
- Response Time: Timeliness also affects one's score. Jurors have to render verdicts within the prescribed period. Voting within that period earns 1 point. Failure to make a decision within the prescribed period results in a null vote, which may affect the case outcome. Overtime decisions, therefore, subtract 2 points from the score, and repeat failures to render timely decisions will result in heavier penalties.
- Appeals: Although no juror can satisfy all expectations and fit everyone's concept of justice, jurors are motivated to render objective and unbiased verdicts. The appellate review, handled by higher-leveled jurors, sufficiently reduces the potential for bias. If the appeal is denied (and the original result is upheld), prevailing jurors' votes will earn an additional 1 point, to reward them for correctly deciding the case outcome. If the original result is overturned, however, former prevailing jurors will lose 3 points, and jurors whose votes align with the appellate result will earn 2 points.
- OATH Token holding: each juror can buy extra OATH token to boost the credit level.

Juror can earn 1 point for every 1,000 OATH tokens they bought. The points earned by buying OATH token will be vested in a 20-day vesting period, linearly. For instance, juror A bought 1,000 OATH tokens (tx1) on day-1, another 1,000 OATH tokens (tx2) on day-2 and sold 1,000 OATH tokens (tx3) on day-3

Day	1	2	3	...	20	21	22	23
Pt / tx1	0.05	0.10	0.15	...	1.00	1.00	1.00	1.00
Pt / tx2	0	0.05	0.10	...	0.95	1.00	1.00	1.00
Pt / tx3	0	0	-0.05	...	-0.90	-0.95	-1.00	-1.00
Total Pt	0.05	0.15	0.20	...	1.05	1.05	1.00	1.00

Juror and juror selection

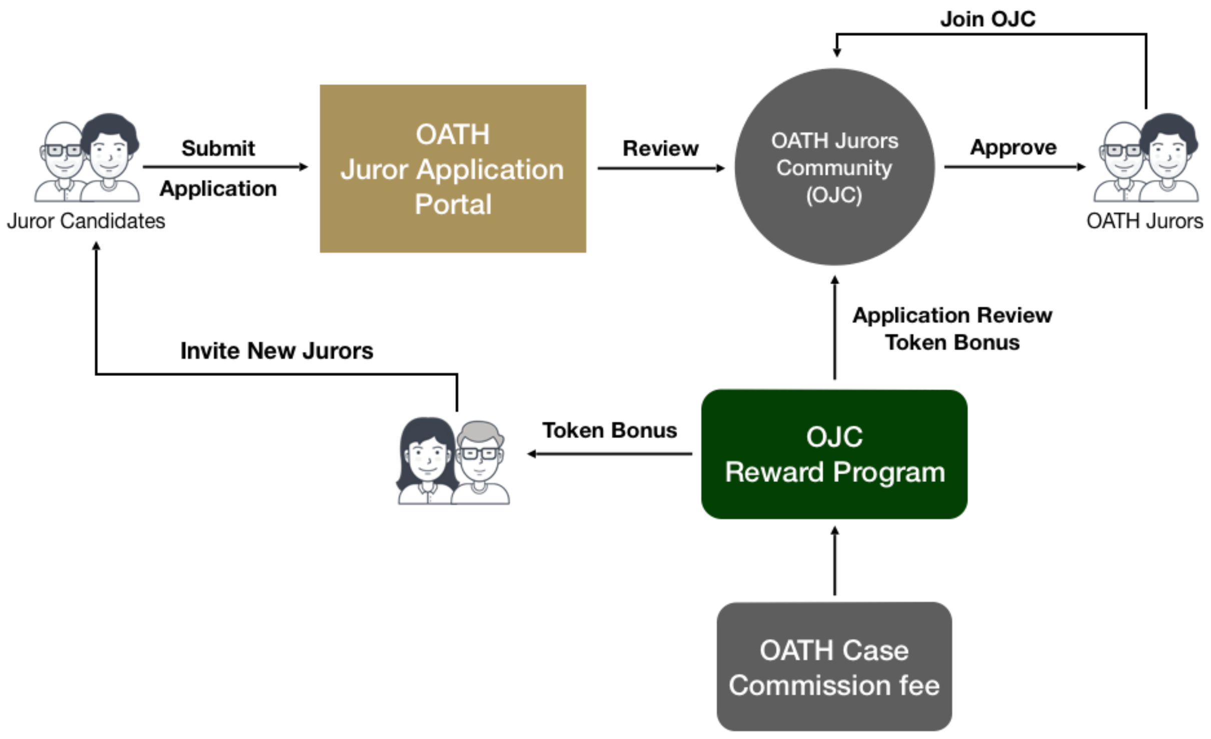

Figure. OATH Juror community

Every juror in the OATH community is assigned Categories and Attributes. Categories contain their personal information and serve as keys for classification and more scattered random screening. Attributes are designed to avoid possible collusion or fraud, and are adjusted based on the juror's case history and the platform development.

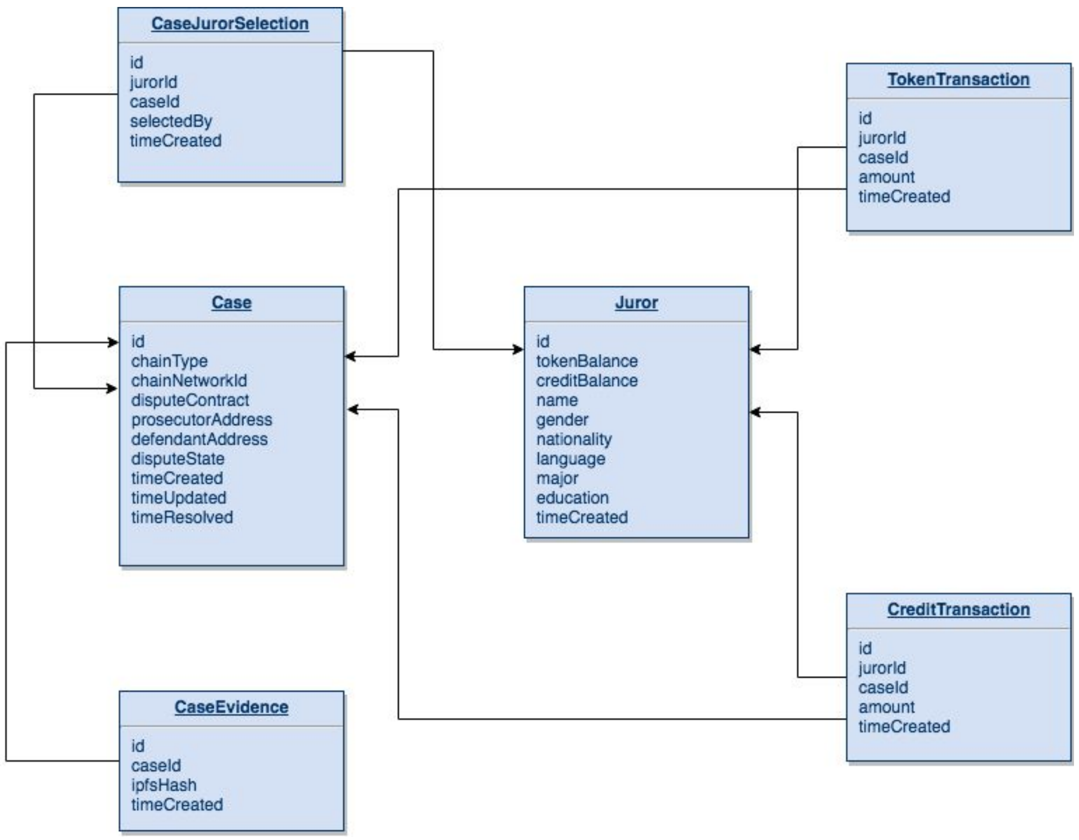

Figure. OPP object relationships

Categories include: age, gender, nationality, language, major, occupation, education, etc. Because fair resolutions require consensus of the general public, OATH will pool jurors from diverse backgrounds to decide each case. Users will provide the relevant category information in their applications. Once the information is submitted, it can be modified via a subsequent application that requires approval by the juror community.

Attributes reflect the map of dynamic relationships between jurors, tracking whether any two jurors have participated in the same case. OATH will record this collaboration relationship and prioritize jurors with no recent attributes in recent cases. This map is constantly adjusted, avoiding potential collusion and fraud that may to occur in continuous groups.

OathProtocolContract (OPC) provides function to integrated DApp to set the juror selection criteria for both parties in dispute. After receiving the criterias, the modern version of the Fisher-Yates shuffle algorithm will be used to randomly choose the jurors for each dispute case.

```
let a := array_jurors
let n := a.length
```

```
struct bucket {
    data;    // juror bucket array
    func;    // function returns true if a given juror meets the criteria
}

// Phase-1 Fish-Yates shuffle
for i from n-1 downto 1 do
    j = random integer such that 0 <= j <= i
    exchange a[j] and a[i]

// Phase-2 Bucket filling
let fulfilled := false
let bucket_1 := new bucket() // Jurors filtered by criterias set by Party-A
let bucket_2 := new bucket() // Jurors filtered by criterias set by Party-B
let bucket_3 := new bucket() // Jurors chosen by OATH platform
for i from 0 up to n-1 do
    if bucket_1.is_full() and bucket_2.is_full() and bucket_3.is_full()
        fulfilled = true
        break
    if (not bucket_1.is_full()) and bucket_1.func(a[i])
        bucket_1.data.append(a[i])
    else if (not bucket_2.is_full()) and bucket_2.func(a[i])
        bucket_2.data.append(a[i])
    else if (not bucket_3.is_full())
        bucket_3.data.append(a[i])

// Phase-3 final check
if not fulfilled
    throw Exception("No sufficient amount of jurors for this election")
```

Note: exception is expected if the criterias set by Party-A and/or Party-B can not be fully met. In such case, OPP will callback the filing contract on this juror selection failed state.

Evidence request and submission

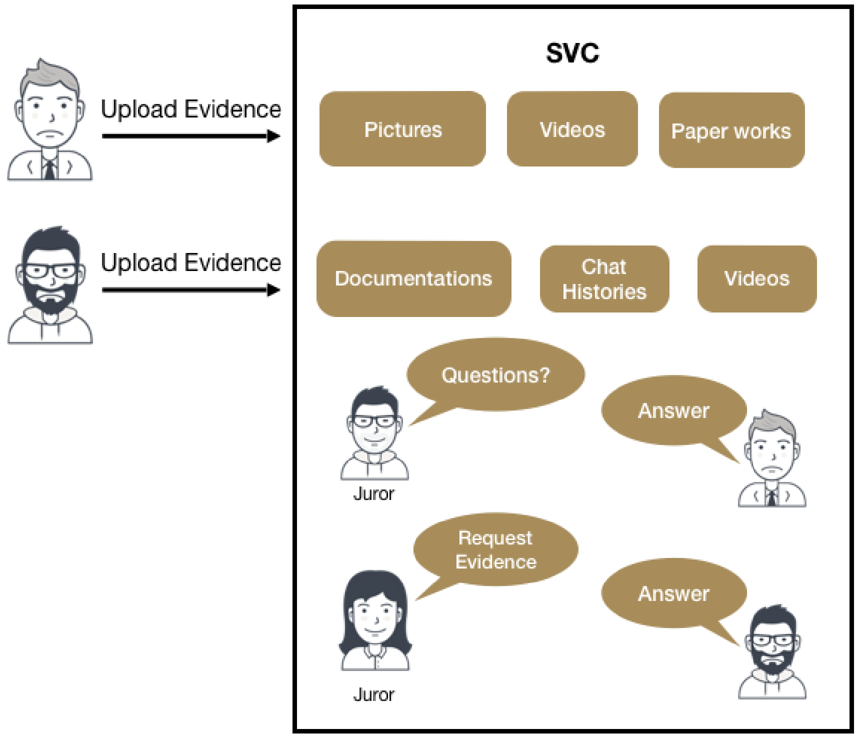

Figure. Dispute evidence submission

OathProtocolContract (OPC) would provide API which allows Party-A and Party-B to submit evidence. All evidences live in IPFS, and the submission API would be

```
contract OathProtocolContract {
    ...
    function submitEvidence(string disputeId, string ipfsHash) external;
    ...
}
```

While the OATH application would surface the evidence in a more user-friendly form (such as image, PDF document and etc.) to jurors. At the time being, we encourage both Party-A and Party-B to be part of OATH community to receive the request for evidence.

Follow the money[4]

GAS

Based on the workflow, following steps are GAS involved:

- DApp gets OPA instance in current network by querying OPR
- DApp creates dispute via OPA
- OPA callbacks onDisputeCreated to DApp
- OPA callbacks onDisputeResolved to DApp

OATH would require DApp deposits sufficient amount of fee to cover the whole process when creating a dispute.

OATH Token

Overall, OATH Protocol offers a fixed amount of total supply among all public chains. Partner DApps are required to hold certain amount of OATH tokens for arbitration fee. For each dispute, the OATH token deposited by DApp contract on creation would be distributed to selected juror after the case is closed.
For more details, see our post on Medium[5].

Alternatives considered

Kleros-like full decentralized resolution

Kleros is a fully decentralized resolution in a similar domain area to OATH Protocol. Due to its decentralized nature, however, it is extremely difficult, if not impossible, to implement a juror community. For instance, the request and submission of evidence and discussions about a dispute case would result in numerous on-chain transactions, resulting in prohibitively high gas fees associated with recording those transactions.
For more details, see our post on Medium <TODO>

[4] https://en.wikipedia.org/wiki/Follow_the_money
[5] https://medium.com/oathprotocol/the-oath-token-af49760f36f