**Chainlink Verdikta – Smart Contract**

DRAFT

10/15/2024

**Introduction**

This is a project to use an oracle service implemented using Chainlink to allow trustless resolution of disputes with artificial intelligence (AI). It will allow two entities, A and B, say, to enter into an agreement where entity A performs work for B with specified terms, with automatic resolution of the problem of A performing the work and B not paying for it.

Along with the creation of a contract document with a registered CID, the system will require Entity A to post Escrow A and Entity B to post Escrow B. Escrow can be either Ether or an ERC-20 token. After an agreed-to amount of time, the two escrows (or whatever remains of them) will be refunded to the original poster if no action is taken. If A and B agree, arbitrary amounts from Escrow A and Escrow B can be paid at any time to arbitrary addresses. At any time, A can submit evidence of completion of the project through a convenient user interface that places the case—text, which may include hyperlinks and references to blockchain transactions, and photos—in a zip file on IPFS and this is reviewed by an oracle service that returns a score for likelihood of completion. If this score exceeds a pre-set score, Threshold 1, then escrow B is paid immediately to person A. If the score is below a pre-set score, Threshold 2, then entity B can submit an IPFS-enabled case for cheating. If the AI-generated score for this exceeds a pre-set score, Threshold 3, then escrow A and escrow B are paid to entity B.

**Role of the Smart Contract**

This smart contract will allow a separately designed and built user interface to communicate with it to do the following:

1. Two participants, A and B, will be able to post escrow and tie it to a contractual agreement for A to do provable work for B. The contract will define a duration for the contract (after which funds are automatically returned) and thresholds for adjudicating disputes using an AI judge. It can be assumed that A and B will have Ethereum wallets, such as MetaMask or Ledger.
2. It should allow A and B to agree to distribute the funds arbitrarily.
3. It should allow A to enter a dispute by providing proof (such as by giving the actual software, photographs, or artwork) of doing the work described in the contract. If this is compelling to the AI judge, all escrow funds will be paid to A.
4. Should A fail in a dispute, it should allow B to provide proof of fraud and if successful in convincing the AI judge, all escrow funds will be paid to B.

Escrow amounts may be either native Ether or an ERC-20 token, such as a stablecoin.

**Interfaces**

The Smart Contract. It has the following general interfaces:

*Interface with Chainlink Node*

The interface between the Smart Contract and the Chainlink Node that will evaluate the data posted to IPFS and return a score will be defined by Ethereum and Chainlink requirements. This communication will happen over the Ethereum blockchain.

A specialized smart contract affiliated with the Chainlink Node will be used for this purpose, called the Chainlink Node oracle contract. This smart contract is a standard contract available to support fielding chainlink nodes (Oracle.sol). This contract will have an address, called the oracle contract address, that any smart contract calling it will need to know.

The input from the Smart Contract to the Chainlink Node will include the following information: 1) a CID for the data, the oracle contract address, the calling contract address, the Chainlink job ID (a value specific to the AI request call on the Chainlink Node that persists over time), and a fee. The returned information from the Chainlink Node to the Smart Contract will use a callback function contained in the calling Smart Contract. The retuned information will include a vector of integers and a CID corresponding to the textual reasons for the results.

Details

Here is an example—very simple—smart contract that all it does is make a call to the Chainlink Node:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@chainlink/contracts/src/v0.8/ChainlinkClient.sol";

contract AIChainlinkRequest is ChainlinkClient {

using Chainlink for Chainlink.Request;

// The Oracle and job spec

address private oracle;

bytes32 private jobId;

uint256 private fee;

// Result from the AI evaluation

uint256[] public evaluationResult;

string public descriptionCID;

// Constructor sets up Chainlink contract parameters

constructor(address \_oracle, bytes32 \_jobId, uint256 \_fee, address \_link) {

setChainlinkToken(\_link); // Chainlink token contract address

oracle = \_oracle; // Chainlink node oracle contract address

jobId = \_jobId; // Job ID from Chainlink node

fee = \_fee; // Fee for the request (in LINK tokens)

}

// Request AI evaluation via Chainlink node

function requestAIEvaluation(string memory cid) public returns (bytes32 requestId) {

Chainlink.Request memory request = buildChainlinkRequest(jobId, address(this), this.fulfill.selector);

// Send the IPFS CID to the external adapter

request.add("cid", cid);

// Send the request to the Chainlink node

return sendChainlinkRequestTo(oracle, request, fee);

}

// Callback function called by Chainlink node with the AI evaluation results

function fulfill(bytes32 \_requestId, uint256[] memory \_evaluationResult, string memory \_descriptionCID) public recordChainlinkFulfillment(\_requestId) {

evaluationResult = \_evaluationResult;

descriptionCID = \_descriptionCID;

}

}

Future Development

Eventually, an aggregator contract will be used to provide trustlessness by polling multiple Chainlink Nodes and eliminating outliers. This component will be added in the future, and need not be explicit in the current design.

***Interface with the User Interface***

The interface between the User Interface Backend and the Smart Contract will be defined by Ethereum requirements. This communication will happen over the Ethereum blockchain. The User Interface Backend will communicate using tools that are part of the Web3.js JavaScript library. The User Interface Backend will use the third-party service Infura to access the Ethereum (or L2) blockchain.

The User Interface Backend will communicate with the Smart Contract by posting a transaction. The backend will sign the transaction using information provided by the user’s cryptocurrency wallet (e.g. Metamask or Ledger). The parties involved in a contract will run separate user interfaces, and will need to agree to terms separately.

The information provided will, in general, include a vector of CIDs for evaluation. One of these CIDs may point to the original contract, while another may point to a dispute document.

**Technical Specifications**

The Smart Contract will be designed to run both on Ethereum and on the compatible platform BASE. The programming language will be Solidity, in particular Solidity 0.8.19.

The Smart Contract will need to store the escrow funds and the related CIDs for the contracts. It will need to be able to distribute the funds according to the rules specified above.

The Smart Contract should have reentracy protection using patterns like the checks-effects-interactions pattern.

It should ensure safe arithmetic using libraries like SafeMath if the safe arithmetic is not handled automatically by Solidity.

Code should be commented for clarity.

**Testing and Audits**

Plans and supporting functions should be provided for thoroughly testing the new code. This should include unit tests for individual functions and integration tests. The testing framework can use Truffle.

**Documentation**

Provide detailed user documentation for end-users or developers interacting with the smart contract and detailed developer documentation for future developers.

**Q&A**

1. Q: How Are the Thresholds (Threshold 1, Threshold 2, Threshold 3) Determined and Managed? A: The AI will return its assessment as a set of two numbers: a score for true and a score for false. This will answer a question like, should entity A be paid? This can be seen as a likelihood or probability when the score for true is divided by the sum of the scores. For determining payment for work, a threshold of 60% for Threshold 1 might be reasonable, with a Threshold 2 of 30%. Threshold 3 should probably usually be higher, at 80% or so. These thresholds will be configurable in the contract text with defaults of {60%, 30%, and 80%}.
2. Q: How Will IPFS Integration Be Handled for Uploading and Retrieving Evidence? A: This smart contract will not need to worry about the details of what is provided in the zip files indicated by IPFS CIDs. These will be made and posted to IPFS in the front end, with the Chainlink external adapter downloading and using the information from IPFS. The content will never be used—or even accessed—by the smart contract.
3. Q: What Are the Expected Costs and Funding Mechanism for Chainlink Oracle Services? A: Great question, the two users, A and B, will need to provide enough Ethereum and LINK to support the task. Please integrate this payment into the smart contract.
4. Q: How Will Disputes Be Initiated, Managed, and Resolved Beyond AI Evaluation? A: Entities A and B may be able to seek legal recourse beyond the AI system, but this is not guaranteed. For many meaningful applications, there will be no meaningful recourse.
5. Q: What Security Measures Beyond Reentrancy Protection Will Be Implemented? A: Please consider other security measures, such as handing of unexpected inputs, and integrate what would be helpful into the smart contract and procedure. Also, if there are good libraries to help with this (e.g., OpenZeppelin Contracts) in a standardized way, please include them. Include a strategy for managing and updating the contract in response to newly discovered vulnerabilities, such as an upgradeable proxy pattern, e.g. the Transparent Proxy or Universal Upgradeable Proxy Standard (UUPS) that separates the contract's logic from its data storage. This separation allows the contract's logic to be updated without altering the stored data.
6. Q: How Will Fund Distribution Be Managed, Especially in Partial Payment Scenarios? A: A partial (or total) distribution may be requested and honored at any time provided it is signed by both parties. Fund distribution within our system is managed through a structured Request-Approval Pattern that ensures both Entity A and Entity B must mutually agree before any funds are released, including in partial payment scenarios. When either party wishes to distribute funds—whether in full or as a partial payment—they initiate a unique distribution request specifying the desired amount and the recipient address. Each request is assigned a distinct identifier and tracked individually within the smart contract. Both Entity A and Entity B must approve the distribution request by individually confirming their consent. To prevent funds from being locked indefinitely, each distribution request is subject to a predefined timeout period. If both parties approve the request within this timeframe, the specified amount is automatically transferred to the designated recipient. However, if one or both parties fail to approve the request before the timeout expires, the distribution request becomes void, and the funds remain securely in escrow. This mechanism not only facilitates flexible and precise fund management, allowing for partial payments as needed, but also ensures that all distributions are consensual and time-bound, thereby maintaining trust and security between the involved parties. The timeout period will be changeable when agreed by both parties.
7. Q: What User Interface Features Are Essential for Seamless Interaction with the Smart Contract? A: The user interface must support the compilation of documents into a zip file, placing the zip file on IPFS, and using the CID to generate a contract or request for payment.

**Instructions**

Based on this information, assume the role of an expert developer and write the Smart Contract, along with testing instructions and documentation. Please ask me any questions you may have to help with a correct implementation.