ORACLES

Blockchain technologies, lecture 4

Course overview

- Oracles
  - Blockchain middleware, general concepts
  - Oracle's problem: decentralization and security
- Oracles design patterns
- Case study: Chainlink architecture
- Case study: Oraclize or Provable Things

ORACLES

ORACLES access off-chain data

- Third party services that provide smart contracts with external information.
- Broaden the scope in which smart contracts operate.
- Oracles classification:
  - Source: hardware, software or human
    - Software Oracles (deterministic oracles) transmit information from online databases, servers, websites etc.

    - Hardware Oracles: translates real-world events in information that can be understood by smart contracts; sensors; Supply Chain applications

    - Human oracles, cryptographically identified, transmit respond to arbitrary inquires.
  - Direction of information: inbound or outbound
  - Trust: centralized or decentralized

ORACLES provide data and off-chain computations

- Data feeds that connect blockchain to off-chain information.
- Inbound or outbound
    - Inbound oracles transmit information from external sources to smart contracts
        - Temperature measured by a sensor
    - Outbound oracles send information from smart contracts to the external world.
        - Smart lock: outbound oracle receives information from smart contract to unlock the smart lock.

ORACLES provide data and off-chain computations

- Data feeds that connect blockchain to off-chain information.
- Applications:
  - payments;
  - exchange rates, decentralized finance (DeFi) applications;
  - predictions, bets;
  - weather reports (insurance calculations);
  - Output Oracles: IoT sensors for supply chain;
  - Dynamic NFTs, gaming, verifiable randomness (fairly select a winner in a lottery);
  - Insurance smart contracts etc.

ORACLES provide data and off-chain computations

- Data feeds that connect blockchain to off-chain information.
- Scale blockchain: off-chain computations
- Functionalities:
  - Receives requests from smart contracts;
  - Send data to external systems;
  - Extract data;
  - Compute: verifiable RNG, aggregate data etc.;
  - Format data in blockchain compatible formats;
  - Validate: zero-knowledge proofs, Trusted Execution Environment (TEE) attestations, TLS signatures etc.

ORACLE PROBLEM

ORACLES access off-chain data

- Data feeds that connect blockchain to off-chain information.
- Blockchain: deterministic transactions, same result on every node, independent of the moment in time the transaction is processed.
- API calls may result in different results and in the impossibility to reach consensus.

- An oracle records the result of calling an external API through a blockchain transaction (blockchain middleware).
- Typically, an oracle is a smart contract accessing some of-chain components that can query APIs.

ORACLES centralized/decentralized

- Smart-contracts cannot directly interact with data existing outside blockchain environment.
- Centralized oracle: single point of failure.
- Security risks (who controls the API?).
- Blockchain immutability: faulty data cannot be reversed.
- Solutions:
  - Decentralized Oracle (DON – decentralized oracle network) combines multiple data sources and multiple independent oracles.
  - prove data validity

ORACLES centralized/decentralized

- Solutions:
  - Decentralized Oracle (DON – decentralized oracle network) combines multiple data sources and multiple independent oracles.
    - Requires predefined standard data format.
    - Inefficient, gathering answers from multiple sources requires time.
    - Data providers may require fees.
  - prove data validity:
    - No need to trust the oracle
    - Data providers don't have to modify services in order to become compatible with Blockchain technologies.

  - Decentralized oracles distribute trust between many participants.

ORACLES DESIGN PATTERNS

ORACLES centralized/decentralized

- Key functionalities:
  - Collect data from off-chain source
  - Transfer the data on-chain with a signed message.
  - Store the data in a smart contract’s storage.

  - Once the data is available in a smart contract’s storage, it can be accessed via message calls.

  - Immediate-Read oracles:
    - Examples: data about persons/organizations, data issued by organizations.

ORACLE design patterns

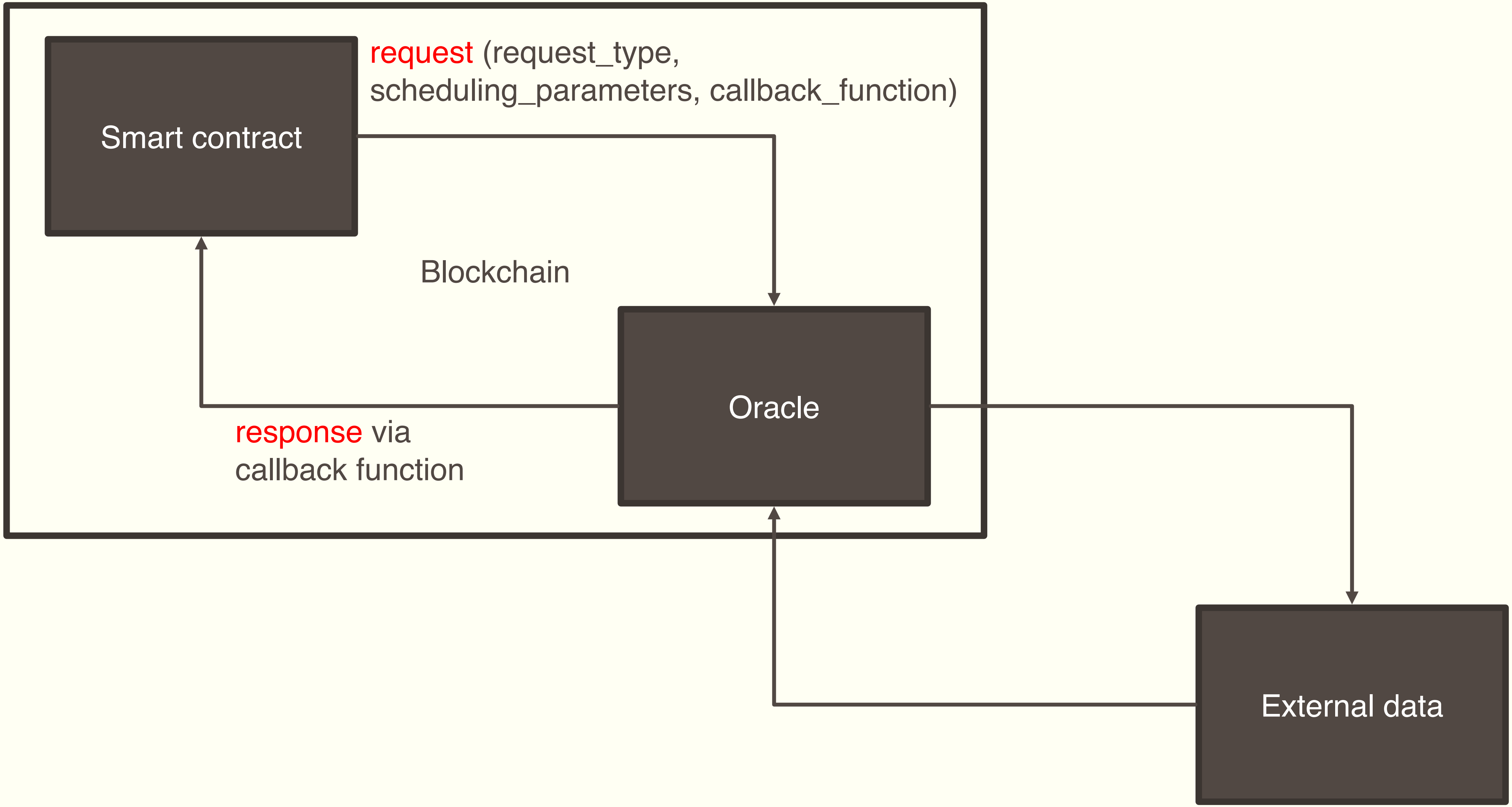
- Publish-subscribe pattern.
- Data is expected to change frequently: prices, weather, traffic data etc.
- Similar to RSS feed.
- Subscribers listen to oracle contracts updates.
- In Ethereum: event logs, can be considered “push” service.
- Publishers categorize messages into classes. Subscribers may express an interest in one or more classes.

ORACLE design patterns

- Request-response pattern.
- Data can’t be stored in a smart contract.
- On-chain smart contract and off-chain infrastructure.
- Requests include scheduling parameters and callback functions.
- The Oracle may require payment for processing the request.

ORACLE design patterns

- Request-response pattern.
- Oracles receives a query from a Dapp with arguments detailing the data requested, callback functions and scheduling parameters.
- Parse the query
- Check payment
- Retrieve data from off-chain source and encrypt it if necessary.
- Broadcast the transaction to the network.
- Schedule any further necessary transactions.


Smart contract
request (request_type, scheduling_parameters, callback_function)
Blockchain
Oracle
response via callback function
External data

ORACLIZE/PROVABLE THINGS

ORACLIZE/Provable Things

- Serves requests on platforms like: Ethereum, R3 Corda, Hyperledger Fabric, EOS etc.
- Operates since 2015.
- Solution to oracle problem: demonstrate data authenticity and validity.
- Data is return together with an authenticity proof. (no need to trust the oracle).
- Authenticity proofs use different technologies, auditable virtual machines and TEE.
- Can be integrated with both private and public instances of blockchain protocols.

ORACLIZE/Provable Things

- Provable engine will execute a given set of instructions if some other given conditions are met.
- “If This Then That” logical model.
- Servs both blockchain-based and non-blockchain-based applications.
- Provable engine ensures a synchronous communication between on-chain smart contract and off-chain external data sources.
- On-chain smart contract executes two transactions:
    - Call request to oraclize engine
    - Callback function call by oraclize engine.

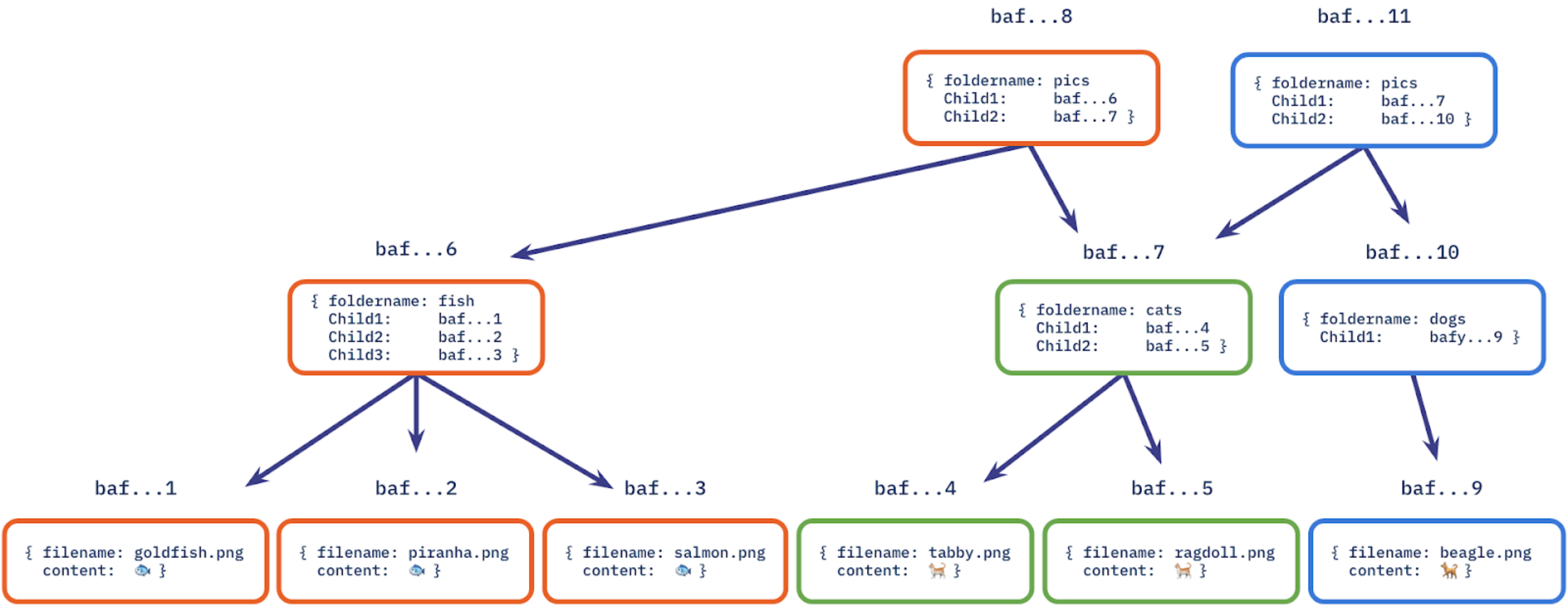
ORACLIZE/Provable Things

- Authenticity proofs:
  - TLSNotary proofs, collection of digital signatures
  - Can be delivered to the smart contract or stored on IPFS.

- Provable data source types: websites or API:
  - URL
  - WolframAlpha
  - IPFS
  - Computation

ORACLIZE/Provable Things

- IPFS
    - P2P storage network
    - Content addressing:
        - identify content and not locations (what you seek instead of where to seek)
        - every content that uses the IPFS protocol has a content identifier (CID) or hash.
    - Uses Merkle DAGs: similar files share parts of Merkle DAG
    - Lookups: Distributed hash table DHT and Kademlia.



baf...8
{ foldername: pics
Child1: baf...6
Child2: baf...7 }
baf...11
{ foldername: pics
Child1: baf...7
Child2: baf...10 }
baf...6
{ foldername: fish
Child1: baf...1
Child2: baf...2
Child3: baf...3 }
baf...7
{ foldername: cats
Child1: baf...4
Child2: baf...5 }
baf...10
{ foldername: dogs
Child1: bafy...9 }
baf...1
{ filename: goldfish.png
content: 🐟 }
baf...2
{ filename: piranha.png
content: 🐟 }
baf...3
{ filename: salmon.png
content: 🐟 }
baf...4
{ filename: tabby.png
content: 🐈 }
baf...5
{ filename: ragdoll.png
content: 🐈 }
baf...9
{ filename: beagle.png
content: 🐕 }

ORACLIZE/Provable Things

- A valid request to Provable should specify the following arguments:
  - Query
  - Data source type
  - Authenticity proof type (optional)

- Query parameters:
  - First parameter mandatory: expected URL where the resources is to be found, if it's the only argument present, the HTTP GET is requested;
  - Optional parameters: data payload of the HTTP POST request;
  - Parsing helpers optional.

ORACLIZE/Provable Things

- Data Source Types: a trusted provider of data, website, web API, a secure application running on a hardware-enforced Trusted Execution Environment (TEE) a virtual machine instance running in a cloud provider etc.

- Native Data Sources:
  - URL enables access to any webpage or HTTP API endpoint
  - WolframAlpha access to WolframAlpha computational engine
  - IPFS access to content stored on IPFS
  - random provides random bytes coming form secure application running on a Ledger Nano S
  - Computation provides result of arbitrary computation

CHAINLINK

Chainlink

- USER-SC Requesting contract.
- CHAINLINK-SC On chain contract responding to USER-SC requests.
- Adapter: interface with CHANLINK-SC on-chain smart and workloads across external services.

Main components:

- Reputation contract:
- Order-matching contract
- Aggregating contract

Chainlink

- Reputation contract: keeps record of the service provider performance metrics.
  - Incentivize and penalize reporting contracts
- Order-matching contract: delivers the request to Chain-link nodes and take their bids on the request and select the number and the types of nodes to fulfill request
- Aggregating contract: calculates weighted answer. The validity of each response is reported to the reputation contract.

- Requesting contract pay with LINK built in accordance with ERC-20 standard.

Bibliography

- https://fravoll.github.io/solidity-patterns/oracle.html
- https://wp.antlia.io/wp-content/uploads/2021/07/Blockchain-oracle-min.jpg
- Abdeljalil Beniiche A study of Blockchain Oracles https://arxiv.org/pdf/2004.07140.pdf
- https://faucets.chain.link/rinkeby
- https://tlsnotary.org/TLSNotary.pdf
- https://docs.provable.xyz/#home
- https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9086815
- https://docs.ipfs.io/concepts/how-ipfs-works/
- https://github.com/tiulia/web3-social-media-dApp/