pairfy

Pairfy,
A P2P ecommerce protocol based on trust and blind pairs.

Juan C. Rey*

November 13, 2023

Abstract. Pairfy is an electronic commerce protocol that uses smart contracts to decentralize functional requirements necessary for the process of selling and acquiring a physical product. Any member of the community can post a product for sale and any member of the community can express an intention to purchase that product. The socket of a product is equal to the stock of a product. If the stock is 15, only the 15 fastest people who express their purchase intention will be able to secure a trading session. A trading session is defined as the process of negotiating a product at a certain time. During the trading session, the buyer and seller generate the negotiation context by engaging in bilateral communication. If the stock is 0 a trading session cannot be opened because there is no socket. A socket can be released if the buyer or seller cancels the session.

1 Introduction

ARKA is an audit protocol that uses smart contracts and a pool of auditors to audit projects in the blockchain ecosystem. Its purpose is to remove centralization of processes in blockchain audits. Auditing companies commonly carry out processes in a centralized manner, assigning auditors, creating reports, editing reports, using private auditing tools and private accounting, among other processes. All centralized processes have limitations to be audited by the ecosystem due to private interests. A decentralized analog audit

*@pairfy website: www.pairfy.io

system solves this problem, processes susceptible to corruption become fully auditable processes.

In *ARKA* the stages of an audit round are *Waiting*, *Voting* and *Auditing.* The protocol uses smart contracts to manage the audit round allowing high auditability by the ecosystem. These three stages are enough to audit multiple projects per round depending on the total budget.

At any time the community can add projects to an auditable list. During the *Voting* stage the *ARKA* holders can vote and request audit services for a project on the auditable list. The result of the *Voting* stage and the availability of the pool of auditors determine how many projects will be audited. Finally in the *Auditing* stage the audits are carried out and the auditors generate the LaTeX reports according to the question scheme selected by voters. There are different types of audits or Q-schemes, the schemes are closed-ended questions with instructions that auditors use to perform an audit and are based on ISO standards. In the current configuration the schemes are *Fundamental Analysis*, *Testing* and *Formal verification.* However it is possible to add more schemes for example a *Chain analysis scheme*, *Rugpull analysis scheme* or *Team doxing scheme.*

ARKA protocol is based on the peer review system. In the *Auditing* stage the active auditors in the pool are randomly assigned in groups of three to the projects. In the current configuration two auditors from the group serves as blind reviewers *Group = [auditor, reviewer, reviewer].* However, it is possible to increase the number of reviewers to improve the quality of the final report. When the report is published the community will be able to review it and earn bounties for making corrections.

The pool of auditors is homogeneous, they all have the same skills to solve any Q-scheme with high quality. It is a necessary condition to be an auditor to have experience making reports in the sandbox. The community and other auditors are in charge of scrutinizing the sandbox reports of the auditor candidates. The community uses the ARKAGOV governance token to elect new auditors after evaluating that there is no reasonable doubt about the abilities and quality of the candidates.

The question schemes used by the auditors are created according to ISO standards. The standards are important to base 3 levels of certification. *Fundamental* (level 1), *Tested* (level 2), *Formal verification* (level 3).

2 Functional requirements

The functional requirements to perform audit rounds are: Round administration, Voting, Random assignment of auditors, Report minting.

2.1 Audit round

A round is a synchronous process of three stages necessary for the audits. Each audit round has a unique name characterized by having the letter R + a consecutive number, e.g. *R1, R2, R3...* The states of an audit round are the *Waiting*, *Voting*, and *Auditing* stages. A stage cannot start if the previous stage has not finished, the transitions are sequential not parallel.

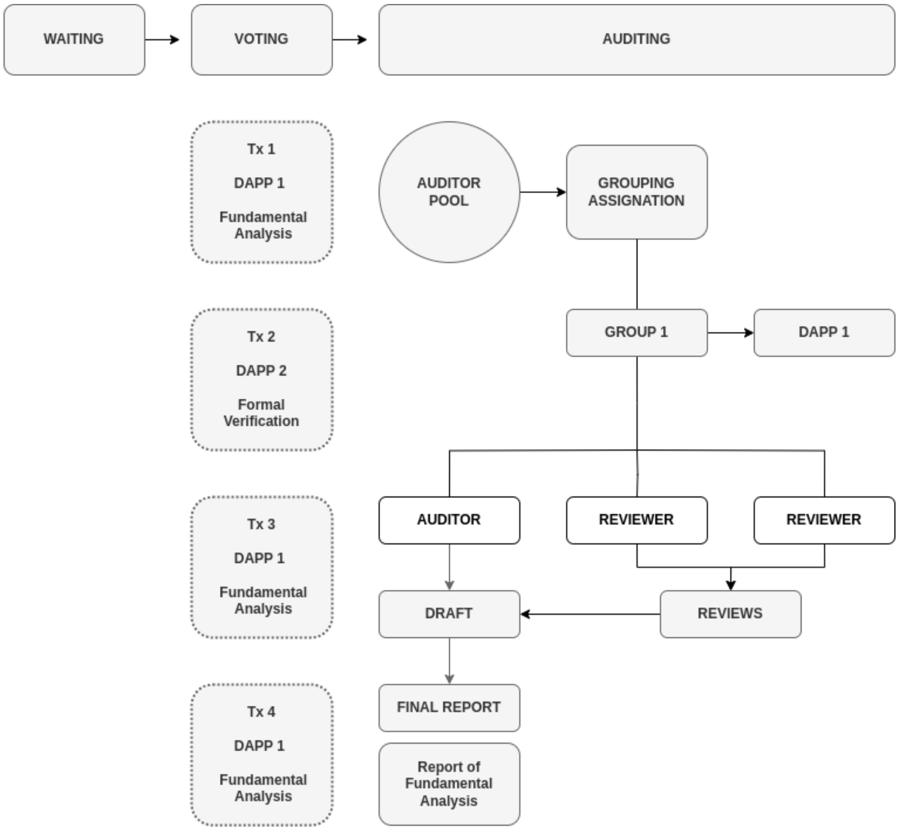

Figure 1: Audit round

In computer science the concepts of state and machine of states are common. A state machine it is a mathematical model to describe the behavior of the different states of a system and their transitions based on conditions, events or triggers. Each state in a state machine represents a specific configuration of the system. It has an initial state that can transition to other states following the rules of the system. Each state within a state machine can execute actions, change variables and produce outputs according to the conditions specifically established for it. There are two types of state machines, the deterministic ones that for a given combination of state and input there is only one possible transition to the next state. And the non-deterministic ones that there can be multiple possible transitions from a given state for a particular input. Fig. 2 shows the deterministic state machine concepts applied to the stages of an audit round. The initial stage is a passive state that does not execute any logic necessary for the audit round in order for the initial state to transition to the first state a trigger is needed. Contracts in Cardano's EUTXO model need at least one initial transaction to trigger their design logic and configure its initial state. In this case the operational wallet interacts with a smart contract endpoint called *startRound* that receives the necessary parameters so that a round can start.

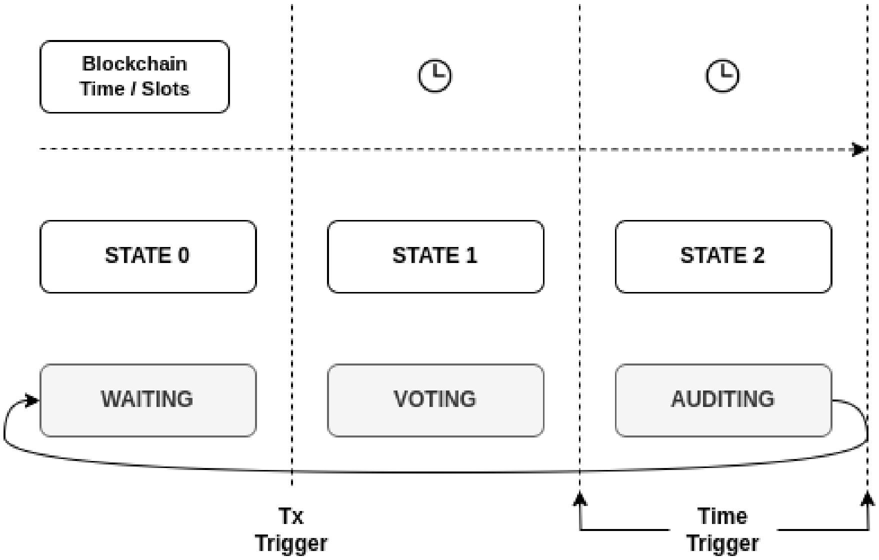

Figure 2: States

```
roundState :: RoundState
roundState = RoundState {
                            mState = 0
                            mLabel = “waiting”
                            vDuration = 0
                            aDuration = 0
                            tProjects = 0
                         }
```

roundState represents the initial state of the smart contract. These variables will remain in the default state indefinitely until the operational wallet interacts with the *startRound* endpoint which initiates an audit round. This is the trigger that makes the contract transition to the first state, that is, the *Voting* stage. Once the operational wallet has interacted with the *startRound* endpoint the contract will transition to the first state by assigning the new parameters to the contract variables.

```
roundState :: RoundState
roundState = RoundState {
                            mState = 1,
                            mLabel = “voting”
                            vDuration = 100
                            aDuration = 100
                            tProjects = 1032
                         }
```

The duration parameters represent the time measured in Slots on the blockchain. The Plutus.Contract module has functions for dealing with time such as waiting for a certain amount of Slots to pass before proceeding with the execution of the contract. It is commonly used when implementing time-based behaviors or waiting for a specific deadline to be reached. It is possible to create a time-based trigger to transition to the second state and also to transition to the initial state without the need for external intervention managed by the time Slots of the blockchain.

2.2 Voting system

2.2.1 State-Snapshot voting system

In the blockchain industry new projects are created daily and the auditable list of projects will inevitably grow over time. It is possible for the community to add 1000 or 10000 projects if they wish. The consequence of this is the large number of indexes in the database. Managing such a number of indexes in a smart contract can be challenging because the limit of Kb per Tx is limited and it is not scalable. However, we can simplify the notion of long-length indices such as those used in databases by using consecutive natural numbers.

A 32-bit unsigned integer can be represented as 0 to 2147483647. A positive integer can be assigned as a unique index to each project added by the community in ARKA. In this way a smart contract could reference a large number of projects using only 32 Bits. For example, an user wants to vote for the project called Minswap which has the index 742 assigned, no other project has this index. The user connects their wallet containing the ARKA utility token to the UI and performs the vote. The request goes to the back-end and contract integration calling the endpoint *createVote* that receives a 32-bit positive integer as a parameter. The contract verifies if the parameter is valid and if the UTxO associated with that wallet address contains the ARKA token. The contract finally checks if the index given as a parameter is less than or equal to *tProjects* variable of the contract which refer to the total number of projects in the auditable list. If these conditions are correct the contract validates the Tx and adds a small mark in the metadata.

Once the *Voting* stage is finished a snapshot is taken at the exact moment or Slot in which the stage ends. By making a query to the blockchain API it is possible to get the transactions associated with the address of the contract to validate the status of the transactions, verify if the transactions have been validated by the contract and verify the metadata of the transaction that provides the context resulting from the interaction with the contract. The metadata can help in identifying the purpose and status of the transaction. The information about the snapshot and governance stage is displayed in the platform UI for all users.

This configuration for the voting system guarantees speed, minimum computing time and the ability to validate millions of indexes using a simple condition:

$indexParam \leq tProjects \Rightarrow \text{True}$. Where *indexParam* is the parameter sent by the user and *tProjects* is the total number of projects on the auditable list.

The parameter *tProjects* can be added by the operational wallet when calling the *startRound* endpoint. This parameter within the smart contract corresponds to a positive integer number. For example, in case there are 1032 projects listed by the community *tProjects* will be 1032. In the initial state of the contract this variable value is 0. At the end of the governance stage this variable will also be 0.

In the hypothetical case that the contract itself was designed to store the

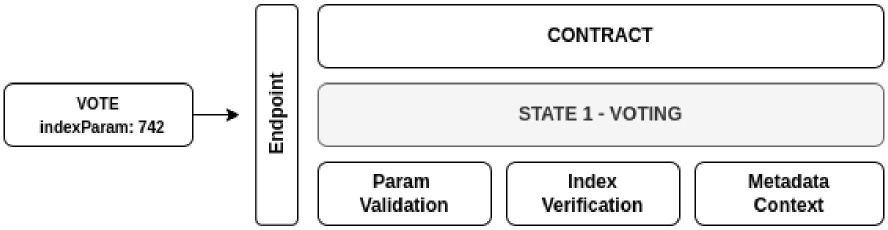

Figure 3: Voting

project indices in the form of assets or NFTs to later be consulted in the governance stage, this would add more logic to the contract and therefore computation time. For this reason it is a disadvantage to use the contract as a form of storage.

However, it is possible to assign a simple time-locked plutus script that allows to store the indices with project names in the form of small metadata using assets (1 asset per project) or simply stamping valid transactions without using assets. The operational wallet is the only one that will be able to interact with this plutus script. The address of the script on the blockchain will need to be included in the metadata when deploying the ARKA contract for the first time for auditability. This solution is scalable since multiple scripts can be used for this purpose. In this way there is complete audability with respect to the indices. Another form of index auditability is public code repositories like Github or distributed storage systems like IPFS. In future iterations of the V1 contract it is possible to add a new stage where the community can add projects to the auditable list using voting.

2.3 Random assignment

Assigning auditors to auditable projects can be a point of low auditability if it is done centrally on private servers. For that reason the best option is a decentralized assignment algorithm. There is not much complexity in the logic required for an equal assignment for all auditors. The main requirements are randomness and uniform distribution of the probability of being chosen as an auditor of a project. The fisher-yates algorithm is a great candidate because it ensures that each element has an equal probability of being placed in any position of the resulting permutation. This is useful since it can shuffle a finite list of indices. For example, A = *[0...50]* where A is the list of indices from auditor 0 to auditor 50. Each index represent a specific auditor and they are ordered consecutively [0,1,2,3,4...50]. When the algorithm is applied to the list the positions of the indices will change randomly.

If ARKA needs 12 auditors for an audit round the first 12 indices from the shuffled list will be selected.

1. *auditorPool* = [0,1,2,3,4...50]
2. *auditorPoolShuffled* = [30, 13, 10, 19, 21, 45, 23, 47, 31, 50, 4, 28, ... 34]
3. *selectedAuditors* = [30, 13, 10, 19, 21, 45, 23, 47, 31, 50, 4, 28]
4. *auditorGroups* = [[30, 13], [10, 19], [21, 45], [23, 47], [31, 50], [4, 28]]

The auditors are randomly selected using the Fisher-Yates algorithm and finally grouped. ARKA requires 2 auditors per project so in this example there are 6 groups for the first 6 projects chosen by the community through voting. The permutations occur on all indexes so there is no need to perform new permutations for role assignment or grouping.

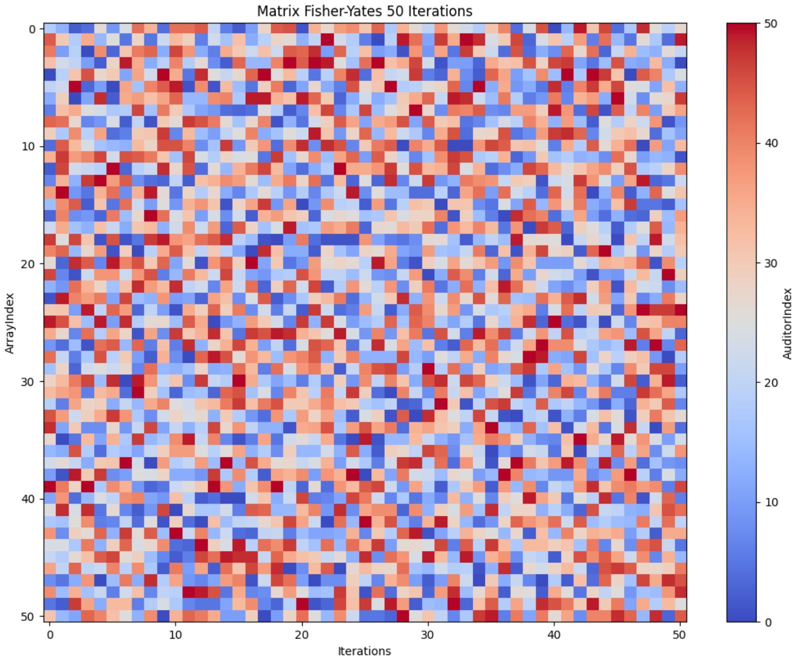

Figure 4: Haskell

Haskell Code

```haskell
{-# LANGUAGE OverloadedStrings #-}
import System.Random (randomRIO)


shuffle :: [Int] -> IO [Int]
shuffle [] = return []
shuffle xs = do
  let m = length xs - 1
  random <- randomRIO (0, m)
  let (left, (chosen:right)) = splitAt random xs
  shuffledRight <- shuffle (left ++ right)
  return (chosen : shuffledRight)
```

This Fisher-Yates haskell version can be used as a reference to create a plutus implementation. The code inside a plutus contract is deterministic it is necessary to use an oracle that generates a random number for the *random* variable or use a pseudo random number generator (PRNG) that takes the hash of the last block generated by the blockchain as a seed of entropy.

2.4 Report Minting

The auditor report and its respective review are two different but necessarily related resources they make up a complete audit report. To ensure the immutability of its content it is necessary mint them as non-fungible assets. This can be done automatically from the backend integration at the end of the *Auditing* stage. The latest version of the .json documents sent by the auditor and the reviewer will be hashed to subsequently mint 3 copies. 1 NFT will be sent to the wallet provided by the auditor. Another will be sent to the reviewer's wallet and another will be stored in a wallet of the DAO. They will be stored in IFPS and Github.

This mechanism can be implemented in the smart contract for its operation during the *Auditing* stage. For example, supplying the contract with the list of wallets that have authorization to mint and some status variables to indicate if they have already minted their report or not. Or use identity tokens as a form of authorization to mint. However this will be the subject of investigation for future versions of the smart contract.

3 Levels of certification

3.1 Fundamental

3.2 Tested

3.3 Formal verification

4 Auditors

The auditors are in charge of creating and reviewing the reports in the *Auditing* stage. They are elected by the community using governance. They have the ability and experience to generate high-quality reports.

4.1 Sandbox

It is a practice environment for potential auditors and amateur researchers. It is a 1:1 workspace used by auditors in audit rounds. Users can conduct research according to the question scheme and generate high-quality reports with the tools provided by LaTeX. The purpose of the sandbox is multiple, train future auditors and provide reports to the community. Anyone can use this environment.

4.2 Designation

The community is in charge of choosing the auditors using the *ARKA* governance token. Anyone can apply if they have previously made 3 reports in the sandbox. The election is completely decentralized.

4.3 Pool

The auditor pool is a unique list of official *ARKA* auditors. An auditor has 2 states, available and not available. Available auditors are those who have expressed interest in participating in the current audit round. Only auditors that have been declared as available can be assigned to auditable projects depending on the random assignment algorithm of the smart contract. A function of the algorithm randomly selects the available auditors and creates groups. *Group = [auditor, reviewer]*. The groups are assigned to the projects previously voted on in *Voting* stage. After this process, everything is ready for the *Auditing* stage to start.

5 Tokenomics