
pairfy

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

Juan C. Rey*

November 16, 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. If the stock of a product is 15, only the 15 fastest people who express their intention to buy will be able to occupy a slot and secure a negotiation session. The slot of a product is a concept that represents the availability to open a negotiation session similar to a sell order on a DEX. A slot can be released if the buyer or seller cancels the session. A negotiation session is defined as the process of negotiation a product for a limited time. The buyer and seller generate the negotiation context by engaging in bilateral communication.

1 Introduction

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. If the stock of a product is 15, only the 15 fastest people who express their intention to buy will be able to occupy a slot and

*@pairfy website: `www.pairfy.io`

secure a negotiation session. The slot of a product is a concept that represents the availability to open a negotiation session similar to a sell order on a DEX. A slot can be released if the buyer or seller cancels the session. A negotiation session is defined as the process of trading a product for a limited time. The buyer and seller generate the negotiation context by engaging in bilateral communication.

2 Requirements

The functional requirements to perform audit rounds are: Negotiation session.

2.1 Negotiation session

The negotiation session is a 4-stage synchronous process *Waiting*, *Locking*, *Delivery*, *Finish*. A stage cannot start if the previous stage has not finished, the transitions are sequential not parallel.

2.1.1 Waiting

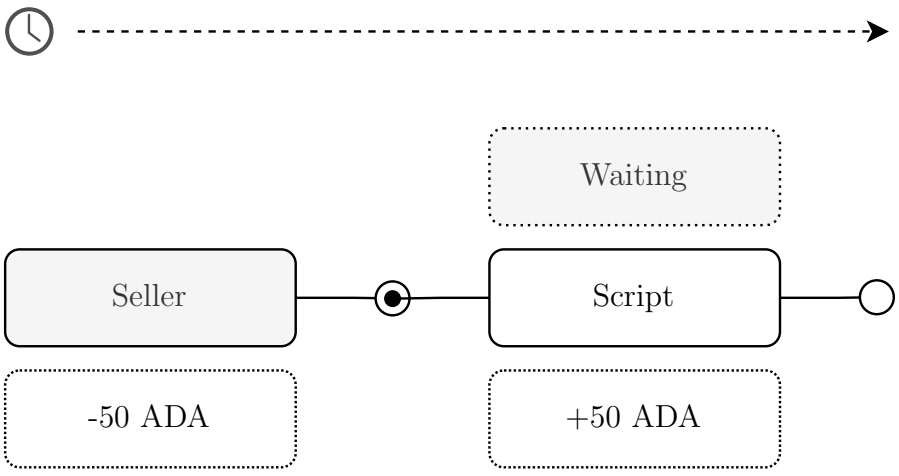

Figure 1: Script deployment

When a seller offers a product to the public a small plutus script with a state machine is deployed on the blockchain. The seller must lock an amount greater than 0 ADA as collateral. If the seller acts in bad faith during

the negotiation session the seller will lose the collateral. If the seller is a good agent during the negotiation session the collateral will return to him. Collateral guarantees that the seller behaves honestly. This mechanism of coercion allows to generate trust in potential buyers. It also allows the seller to increase their trust rating. Example. Alice publishes a book with 20 units of stock. She activates only 10 slots (sell orders) each with a collateral of 20 ADA. In the first few hours 7 books were sold and now there are only 3 slots left. Alice decides to activate the 10 remaining slots due to demand. Bob sells the same product with the same stock but offers collateral of 5 ADA. The community prefers to buy the books from Alice since she is more committed to fulfilling her obligation by offering 20 ADA collateral. Other factors build a seller's trust in buyers such as the total number of successful sales, seniority, or profile information.

In the background for each activated slot, individual scripts are deployed on the blockchain waiting to be occupied by buyers.

2.1.2 Locking

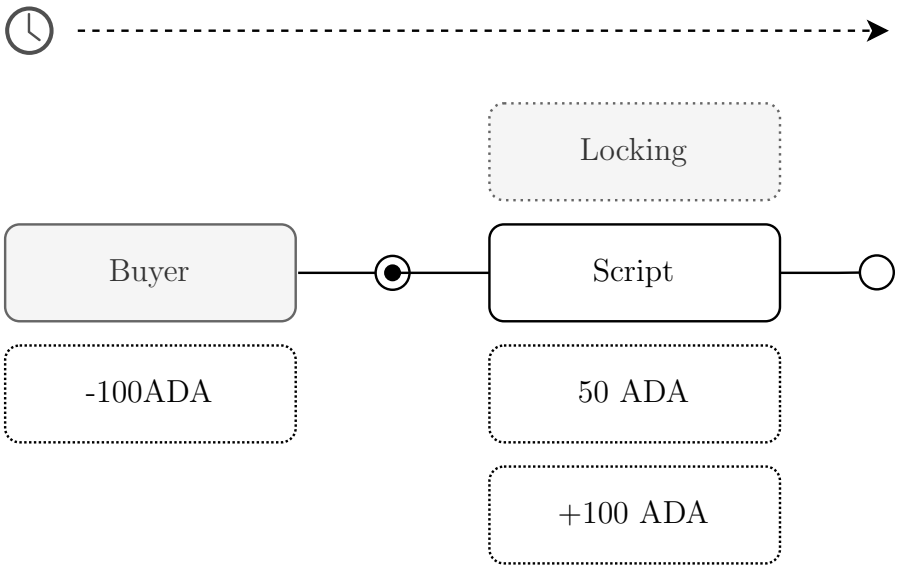

Figure 2: Session locked

In Figure 2 you can see the collateral of 50 ADA given by the seller and the price of the product 100 ADA given by the buyer. The current state of the script is Locking. When the buyer presses the buy button a slot is occupied and the state of the script transitions from *Waiting* to *Locking*. Blocking

funds allows participants to advance their obligations. The seller's obligation is to deliver the correct product with the correct specifications. The buyer's obligation is to receive the product and pay the price. It is important to clarify that the buyer's obligation to pay the price is guaranteed when he occupies a slot since the amount in ADA of the product price is a necessary condition to occupy a slot.

From this point the seller can start with questions such as: What is the delivery point? Description of delivery point? Any questions necessary to guarantee the effective delivery of the product.

All information provided by both actors in the user interface is contained within a websocket instance and can only be observed by the blind peers in case of a dispute. The websocket server does not store any type of information about the negotiation session once legal security is declared about the business between the parties. Legal security is declared by the blind peers and refers to the absence of reasonable doubt in the fulfillment of the obligations corresponding to the buyer and the seller.

2.1.3 Delivery

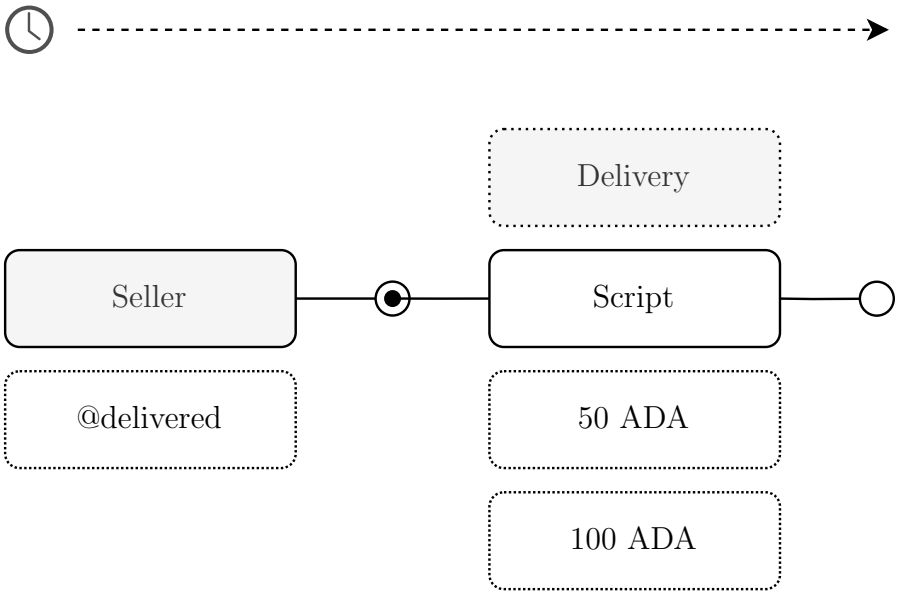

Figure 3: Delivery confirmed by seller

When the shipping company confirms the effective delivery of the product the seller can invoke the @delivered endpoint. The script transitions to

the *Delivery* state which indicates that the seller has fulfilled its delivery obligation. Finally, the buyer confirms whether he received the product by invoking the @received endpoint. By doing so, the script transitions to the last state *Finish* which releases the funds to the seller's wallet.

2.1.4 Finish

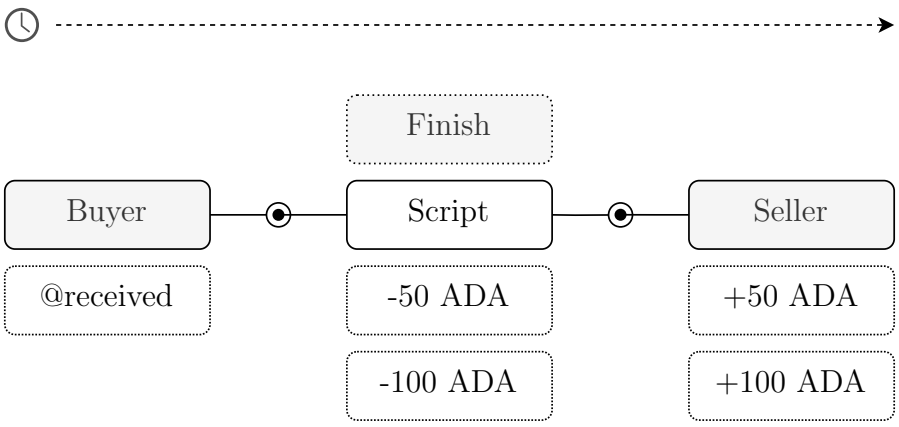

Figure 4: Delivery confirmed by buyer

2.2 State machine

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. 5 shows the deterministic state machine concepts applied to the stages of a negotiation session. Contracts in Cardano's EUTXO model need at least one initial transaction to trigger their design logic and configure its initial state. When the seller activates a slot his wallet deploys a plutus script

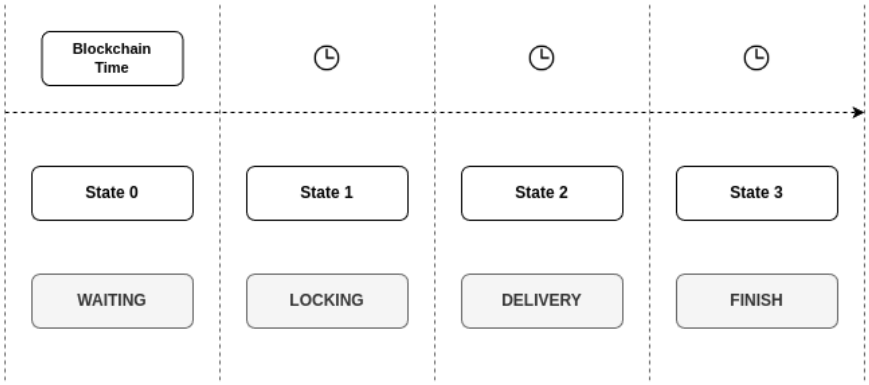

Figure 5: Session states

that receives the collateral in ADA and necessary parameters for the session.

sessionState :: *SessionState*
sessionState = SessionState {
cState = 0
sLabel = “waiting”
tDuration = 100
cSlot = False
pDelivered = False
pReceived = False
}

The data type *sessionState* represents the initial state of the plutus script once deployed by the seller. These variables will remain in the default state indefinitely until the buyer's wallet interacts with the script taking the slot.

sessionState :: *SessionState*
sessionState = SessionState {
cState = 1
sLabel = “locking”
tDuration = 100
cSlot = True
pDelivered = False
pReceived = False
}

cSlot represents the variable that indicates whether the script has been occupied by a buyer. The boolean value True means that the slot has been

occupied by a buyer which makes the script transition from *Waiting* to *Locking*.

```
sessionState :: SessionState
sessionState = SessionState {
                                cState = 2
                                sLabel = "delivery"
                                tDuration = 100
                                cSlot = True
                                pDelivered = True
                                pReceived = False
                              }
```

pDelivered is a variable of type boolean that represents the delivery of the product or not. At this point the seller has invoked the @delivered endpoint stating that he has fulfilled his obligation to deliver the product. This action makes the script transition from *Locking* to *Delivery*

```
sessionState :: SessionState
sessionState = SessionState {
                                cState = 3
                                sLabel = "finish"
                                tDuration = 100
                                cSlot = True
                                pDelivered = True
                                pReceived = True
                              }
```

pReceived is a boolean variable that represents whether the buyer confirms receipt of the product or not. This variable is modified only by the buyer using the @received endpoint. By doing this, the script transitions to its final state *Finish* which releases the funds to the seller.

2.3 Blind pairs

2.3.1 Appeal

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:

$indexParam \leq tProjects \Rightarrow$ 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 project indices in the form of assets or NFTs to later be consulted in the

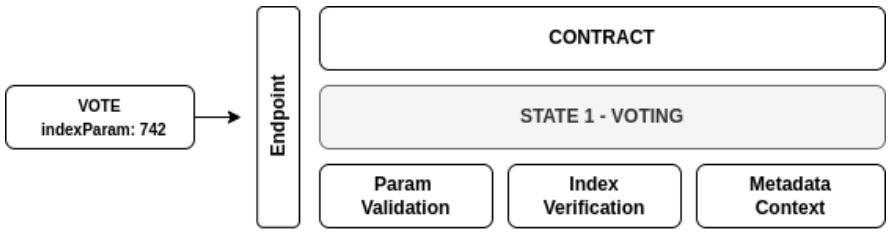

Figure 6: Voting

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.

2.4 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.

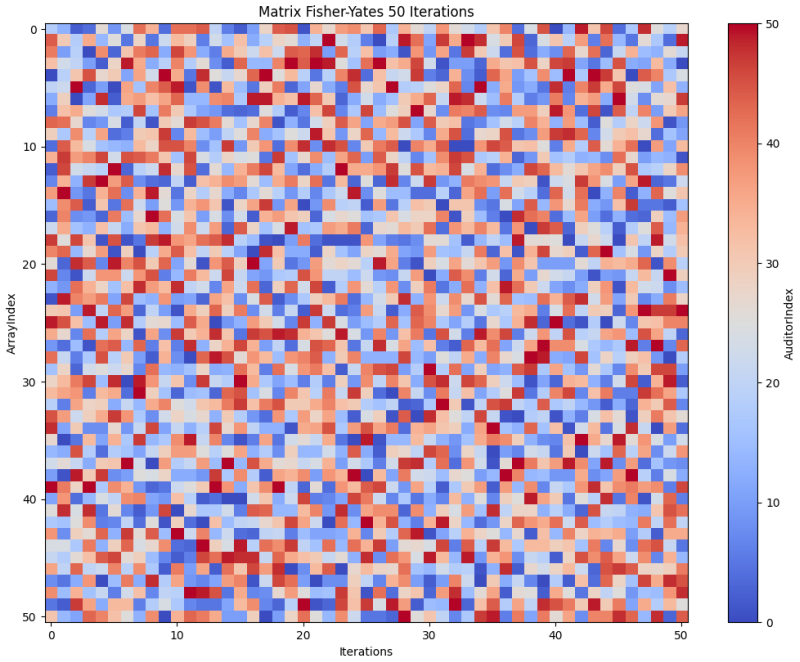

Figure 7: Haskell

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.

2.5 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