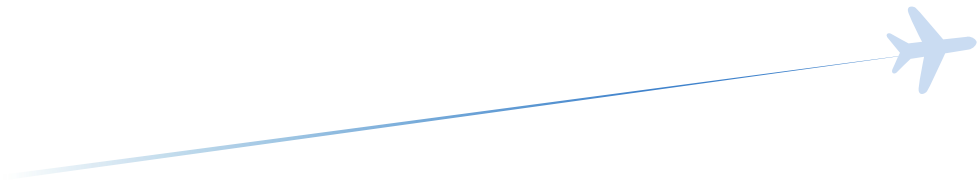

D3.2 Specification of the PrivacyEngine Component

Deliverable ID:	**D3.2**
Dissemination Level:	**PU**
Project Acronym:	**SlotMachine**
Grant:	**890456**
Call:	**H2020-SESAR-2019-2**
Topic:	**SESAR-ER4-27-2019 Future ATM Architecture**
Consortium Coordinator:	**Frequentis**
Edition Date:	**23 December 2021**
Edition:	**01.00.00**
Template Edition:	**02.00.03**

Founding Members
EUROPEAN UNION EUROCONTROL
SESAR JOINT UNDERTAKING

SlotMachine

A PRIVACY-PRESERVING MARKETPLACE FOR SLOT MANAGEMENT

This Report is part of a project that has received funding from the SESAR Joint Undertaking under grant agreement No 890456 under European Union's Horizon 2020 research and innovation programme.

Abstract

In this report we present the architecture developed for the privacy engine and blockchain which are essential components in SlotMachine to achieve security and privacy goals. The privacy engine enables developer-friendly access to multiparty computation, a method to compute on encrypted data, and the blockchain component is used to run a permissioned distributed ledger with a dedicated application. The document shows implementation details about the functioning and inner workings of the different (sub-)systems as well as descriptions of important interfaces for both, external and internal ones. Additionally, we give some design rationale to support our decisions and show positive and negative research results achieved on the way. Moreover, we show intermediate results of ongoing research topics, which will not be part of the final demonstrator but will lead to way to the next generation of SlotMachine.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose of the document

In this report we present the developed architecture and implementation approach of the privacy engine and associated components in more detail and give design rationale for the current design. The privacy engine (PE) is responsible for the protection of sensitive information provided by airspace user (AU) in SlotMachine, and it does so by intensive use of cryptographic mechanisms. Additionally, for the PE to work efficiently and to establish the required trust into the platform, we combine it with Blockchain in a seamless way. Finally, we also present recent research results on possible extensions and identify future work to strengthen the trustworthiness of SlotMachine even further. In summary, the report shows the progress made according to the work plan and important design decisions relevant for the final proof-of-concept.

1.2 Scope

This document covers the specification of the privacy engine component which integrates with the platform as described in *D2.2* [1]. It is designed to support security and privacy requirements as specified in *D2.1* [2], among others, and leverages the technologies discussed in *D3.1* [3]. Along with the PE component we also describe the use of Blockchain which complements the properties of PE and is essential for our approach. The detailed requirements achieved with PE and BC are also summarized in D2.2 and further discussed in this report. Additionally, we identify possible future extensions and research directions which could lead to an even more secure and decentralized platform.

1.3 Intended readership

This document is intended for both internal and external audiences. Internally it is mainly aimed at the technical team members in WP3 but also for WP4 and WP5. In WP4 the developers of the heuristic optimizers are relying on the functionality and performance of the PE component and in WP5 the partners responsible for deployment and testing find the necessary information to integrate the components to the prototype. However, it could be also a useful resource for other project participants and the public because it provides a comprehensive documentation of our approach and identifies future work and research directions to follow.

1.4 Background

The current architecture presented in this report is a culmination of many discussion and research work done in multiple work packages. First, it was designed to fulfil core requirements regarding security, privacy and transparency as described in *D2.1*. However, the solution has to support the business cases presented in *D2.3* [4] and to integrate with the overall platform specified in *D2.2*. Furthermore, was designed in close cooperation to WP4 and the team working on the optimizer, which is documented in *D4.2* [5]. Nevertheless, a lot of research on the technologies used in the PE and Blockchain component were done *D3.1* to select the most feasible protocols and solutions and many different approaches for the integration with the optimizer component were already researched in the feasibility phase documented in *D4.1* [6].

1.5 Structure of the document and relation to other deliverables

The remainder of the document is optimized to give software engineers easy access to important documentation of the PE component and Blockchain use, but also design decisions made. It comprises the following chapters.

- Chapter 2 gives an architectural overview of the PE engine and deployment considerations. Additionally, it introduces the encoding service needed to pre-process sensitive PE input and shows how Blockchain is integrated.
- Chapter 3 documents more inner workings and internal interfaces of the different components to get a better understanding of the expected behaviour and implications for their usage.
- Chapter 4 shows the basis for the design decision made. It shows benchmarks results for alternative solutions which turned out to be not efficient enough but also identify interesting new research directions which could potentially be used to extend the current approach for improved security.
- Based on the current version of the design for the privacy engine and blockchain component we finally conclude in chapter 5 discussing the pros and cons of the current approach. We also give some recommendations for future research directions and identify open problems.

2 Architecture Overview

On a high level the SlotMachine platform combines tools for privacy-preserving computation on data based on multiparty computation (MPC) with evolutionary algorithms and blockchain technology to build a decentralised system that enables collaboration for optimal flight sequencing in challenging conditions. The overall structure is shown in Figure 1. From this approach it becomes clear how the PE component integrates with the rest of the system and as well as the blockchain. The PE is basically holding and managing the sensitive data in encrypted form and assisting the heuristic optimizer. The Blockchain is used to (publicly and immutably) record important data and maintain the credit wallets used in some market models. A more detailed specification of the SlotMachine architecture including relevant interfaces can be found in *D2.2*.

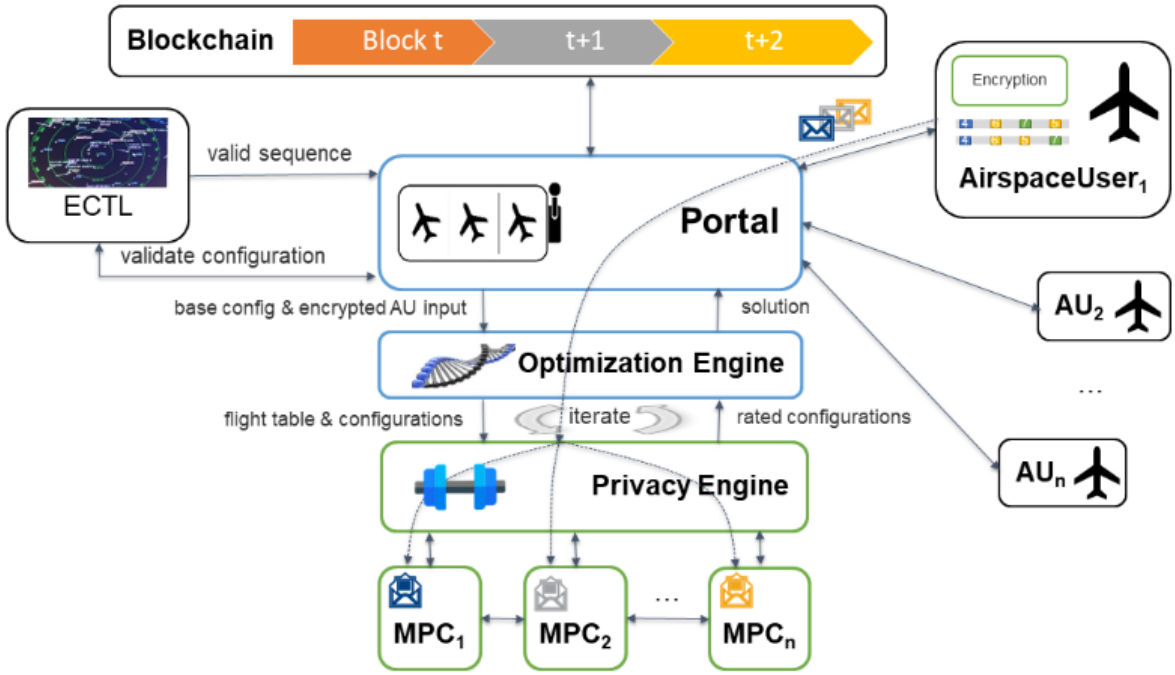

Figure 1: High-Level SlotMachine Architecture with main components.

From a software engineering perspective, the Privacy Engine (PE) encapsulates all complex cryptographic tasks in an easy-to-use manner from the rest of the platform and represents the (distributed) place where sensitive information is managed, i.e., specifically confidentiality is protected. Technically, the Privacy Engine is a module leveraging multiparty computation to process sensitive information in encrypted form only. If information is only processed in encrypted form highest security and privacy standards can be realized.

Additionally, to combine both technologies — MPC and blockchain — in a fruitful way, we developed a dedicated blockchain component which can be easily used over a REST API. The blockchain will be used to store data and to manage credit wallets.

2.1 Privacy Engine Internal Structure

The Privacy Engine consists of the controller and several MPC nodes. The controller presents a simple REST interface and controls the nodes via a basic TCP protocol. In addition, the nodes maintain separate TCP connections with each other, operated and controlled by the underlying MPC framework. Additionally, an encoding service is integrated, which enables users a simple way to encode input data before sending it to the controller (via other components). The encoder can also be run locally on the client side if needed.

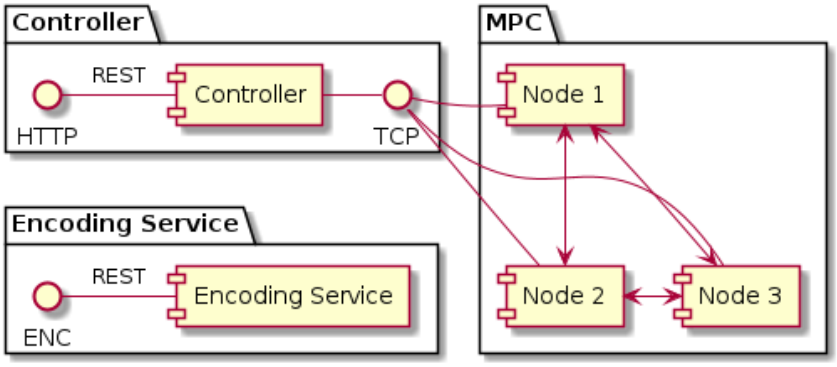

Figure 2: Internal structure of privacy engine with controller, MPC nodes and encoding service.

The main functionality of the Privacy Engine is managing sensitive data. One the one hand, it enables the optimizer to compute the relative fitness of populations without revealing the underlying inputs, i.e., the margins, weights and credits set. That is, after the weight maps have been installed, the optimizer can then compute aggregates on them in a privacy-preserving way. On the other hand, it is responsible for the computation of the clearing once the final flight sequence is selected. Thirdly, the integrated encoding service can be used by the clients to encode and encrypt input data before sending it to the PE over the platform. If data the data is encoded and securely sent to the MPC nodes, it is guaranteed that no component of the platform has access to the sensitive input data of the AUs. The encoding service can be either used as part of the PE or as local standalone service, depending on the needs of the AUs.

For research and debug purposes, the PE also supports a non-privacy-preserving mode, where the encoder component is by-passed, and the controller receives the plain-text weight-map directly. However, the weight-map is still secret-shared and passed on to the MPC nodes as in the fully privacy-preserving scenario. This mode can be safely removed in production environments.

To better understand how the PE should be used we present typical usage patterns. The main idea of the PE is to operate on a session basis. An optimization session belongs to a particular slot swapping session and is defined by the input of the AUs for the current optimization process. In the following, we present the privacy-preserving session for normal operation and a non-privacy preserving session which was mainly implemented for research and debugging purposes. Finally, we quickly present the idea of the encoding service and software packages used.

Privacy-preserving Session

First, the weight-map — a quadratic matrix, expressed as a list of lists of integer values — is sent to the Encoding Service's `PUT /` method that returns three separate weight-maps of the same shape (lists of lists of integer values) that are contained in a map/dictionary that assigns each secret-shared weight-map to an MPC node (hard-coded as "A", "B", "C").

Because each secret-shared weight-map is additionally encrypted with the public key of the MPC node for which it is intended, the Controller does not learn the contents of the original plain-text weight-map. This map can then be given to the Controller's `PUT /sessionSecret` method that passes the secret-shared weight-maps to the respective MPC nodes.

If there were no errors, the Controller's `PUT /computePopulationOrder` method can then compute for a given population of configurations the maximum encountered fitness and a sorted list of indices of configurations, so that the first element of this list contains the index of the configuration that had the highest fitness value, and the last element contains the one with lowest fitness value.

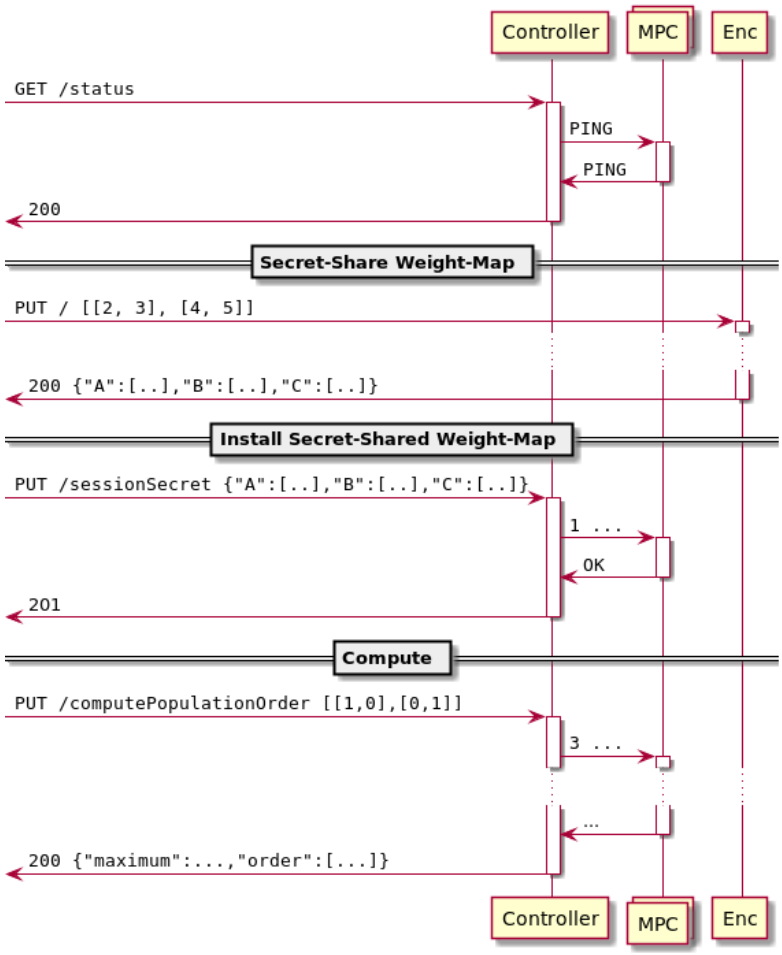

Figure 3: Basic usage of PE in normal privacy preserving mode.

Non-privacy-preserving Session

In a non-privacy-preserving session, the encoding step is skipped and the Controller's `PUT /sessionClear` method takes a plain-text weight map. The secret-sharing step is done in the controller itself. Every step after that remains the same. This means that the Controller can use the MPC backend for all computations and does not need to implement separate non-privacy-preserving methods other than the one to install a weight-map.

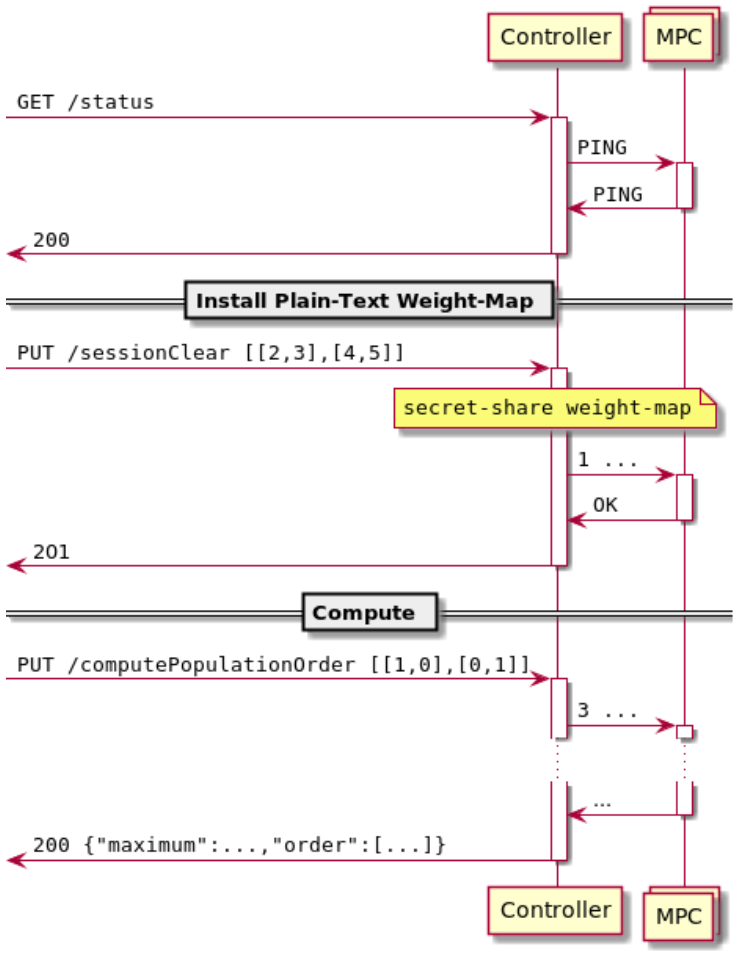

Figure 4: PE usage in non-privacy preserving mode, which is intended for testing.

Encoding Service

Technically, the encoding service secret shares the weight maps for flight prioritization. It turns a plain-text weight map into a secret shared form suitable as input for the MPC nodes. Therefore, it transforms a matrix of plain-text integer weights into three separate matrices, each of the same shape as the original matrix and containing a share of the original plain-text value. The functionality is offered via a single REST method. Additionally, it can also be used by the clients locally and run as a simple stand-alone component.

Implementation

The PE implementation is built on several Python libraries already discussed in *D3.1*: for MPC, the *MPyC* framework is used. The REST interface and automatically generated online documentation is provided by *FastAPI*. *PyInstaller* is used to pre-package the code together with all necessary libraries and the Python interpreter, so that the components can be put into compact Docker containers based on the base Alpine Linux image. As a result, the containers (encoder, controller, MPC node) are only around 13 MB in size each.

2.2 Structure of the Blockchain System

There are several different blockchain ecosystems. We have decided to use Tendermint, because it's well suited for small up to medium sized networks and has a high transaction throughput. Every node which is a part of the blockchain network has to run two software components: the Tendermint client (Tendermint Core) and a blockchain application. These components communicate via a standardized interface (ABCI) while the Tendermint clients on the different nodes communicate with each other and form the consensus engine. The blockchain application contains the actual logic specific to the desired functionality. This is comparable to what is often referred to as a smart contract, with the difference that the code is not put on the blockchain but runs alongside. One benefit of this design is that the code is allowed to be of a higher size and to be computationally more intensive. At every time, at least more than two thirds of the nodes have to be honest and working properly. This means that if we want to allow for one dishonest or malfunctioning node, we need to have at least four nodes in total, which is exactly the number of nodes we will run for the proof-of-concept. In a production phase, ideally each AU would operate one blockchain node, but this is not strictly necessary. Alternatively, an AU A that does not want to operate a blockchain node can cooperate with another AU that runs a node and is willing to grant A access. However, in *D3.1* it has been shown, that the solutions scale to the required size even if all AUs operate a Tendermint node.

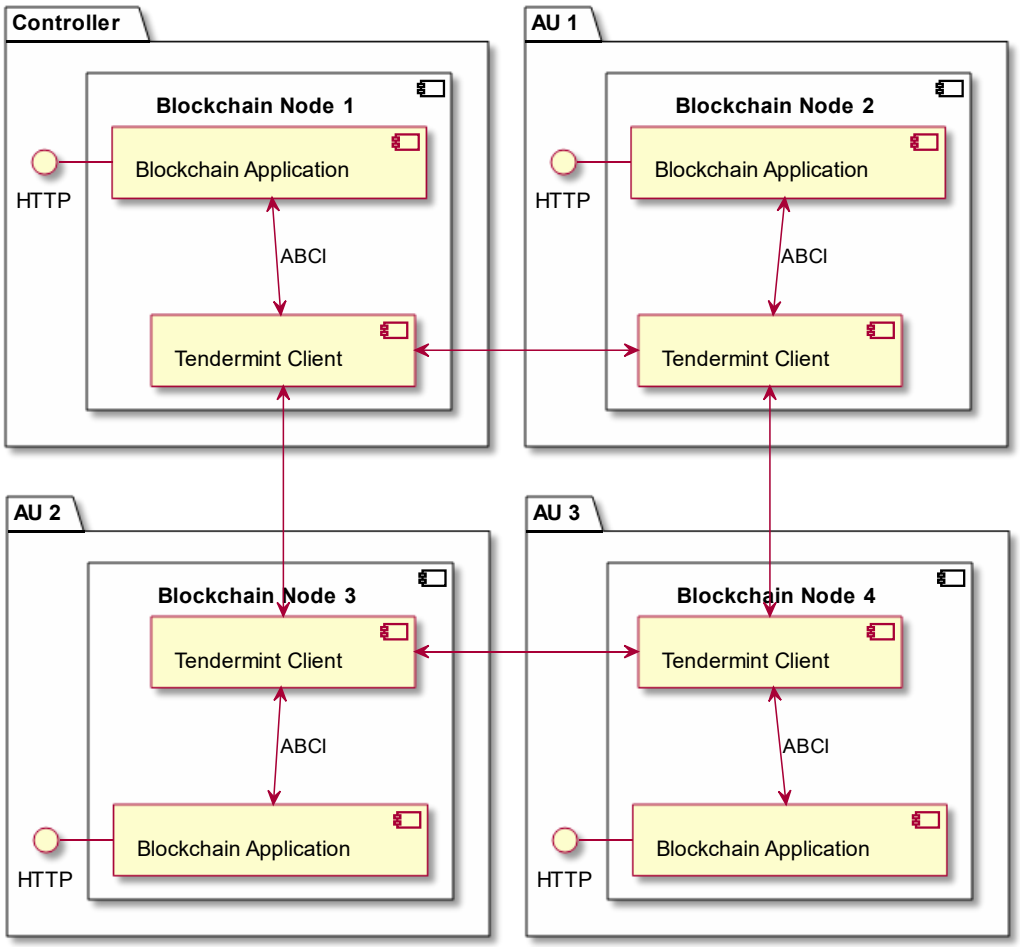

Figure 5: Structure of Blockchain component with 4 nodes as used in first prototype.

2.3 Deployment and Test Strategy

In general, we aimed at the simplest possible strategy to make deployment as easy as possible. Therefore, a container approach was chosen for both, the privacy engine and the blockchain.

2.3.1 Privacy Engine

Each component (controller, MPC node, encoding service) is built as a separate Docker container. The encoding service can also be used independently and has no dependencies on the other services whatsoever and does not need any further configuration. The controller and MPC services need to know the fixed addresses of their connection peers. Thus, the controller must know the addresses of all MPC nodes and the MPC nodes need to know the addresses of each other. This is accomplished by using environment variables to pass information to the container processes. For simple start-up and testing we also provide a docker-compose configuration which deploys a minimal but fully functional configuration. Additionally, we provide test scripts for standalone component testing. The tests cover all major steps and most important failure cases and demonstrate usage scenarios for developers.

2.3.2 Blockchain

There will be either one or two container images for the blockchain system, depending on whether it is practical to split blockchain application and Tendermint client, or not. Conceptually, it would be preferred to have two separate container images, but this might render the start-up procedure more complicated. As the prototype with blockchain will consist of four blockchain nodes, each image is intended to be run with four instances. Nevertheless, it will be possible to change the number of nodes arbitrarily with only a small configuration effort. Wherever it is possible, the capabilities of the Cosmos SDK will be leveraged to provide a small set of automated tests. As blockchain environments like Tendermint often do not provide a very sophisticated support for automated testing, we will test some parts of the functionality manually.

2.4 Relation to SlotMachine Requirements

The PE architecture was developed to satisfy the security and privacy needs for the SM platform as specified in *D2.1*. To better understand how the PE and BC design helps to address important SM requirements we explicitly discuss them in this section. We focus on the relevant privacy and security requirements as they are basically provided by the privacy engine.

ID	Description	Addressed by PE and BC
priv_1	The data provided by the AU and identified as sensitive shall remain protected.	During the curse of the project, AU input of margins and priorities defined by weights and/or credits per slots were identified as sensitive. All this data is kept encrypted throughout the whole process and never handled in plaintext.
priv_2	The AU flight prioritization preferences shall remain confidential and secured from competitors.	See priv_1.
priv_3	The AU flight prioritization preferences shall remain confidential and protected from honest-but-curious platform operator individuals.	Because data is encrypted during processing and because the inputs are already encoded by the client and only sent in encrypted form to the MPC nodes, the platform learns nothing about the inputs except what it can infer from the PE output.
priv_3.1	The AU flight prioritization preferences shall remain confidential and protected from Network Management Functions (incl. Flow Management Positions).	See priv_1.
priv_4	The AU flight prioritization preferences shall be processed in encrypted/encoded form only in the platform.	PE uses MPC to manage and process AU input and never reconstructs/decodes the plaintext. After session is finished encrypted session input is deleted.

ID	Description	Addressed by PE and BC
priv_5	The internally used representation of AU flight prioritization by SlotMachine shall be securely derived from AU input for each flight prioritization and flight considered (e.g., weights).	In the current version of SM1 and SM3 weight maps are used to represent AU priorities. They can be automatically derived from margins or freely configured by AUs. In both cases they are computed locally on the AU side (client) and never sent to the platform in plaintext.
priv_6	Incorrect input data from AUs shall be detectable, e.g., invalid margins and weight configurations.	To verify that encrypted input sent to the platform is valid we offer two techniques. In a first step input is validated directly in the PE and an error is raised if malicious behaviour is detected. Additionally, we are researching ways to offload the task by using NIZK. With NIZK each AU generate a proof that their input is correct which can be efficiently verified by each MPC node.
priv_8	Security shall be maintained against honest-but-curious behaviour of Privacy Engine service operators.	Given by input privacy property of MPC and the fact that all operations are conducted on ciphertext. Protocols used in PE and proper deployment fully support privacy.
priv_9	Security shall be maintained against honest-but-curious behaviour MPC nodes.	Because we are able to do all computations in the encrypted domain and do not need any intermediate steps in plaintext, it is given by the properties of MPC protocols.
priv_10	Security shall be maintained against malicious behaviour of AUs.	By verifying AU input, we prevent malicious behaviour (see priv_6). Additionally, because the optimization does not need any interaction with AU clients, they cannot block the protocol run.
priv_11	The credits allocated by each airline to its flights must remain private.	The credits are also encrypted together with the weight in all supported market mechanisms. Also, for clearing the PE does not reveal any individual inputs.
priv_12	The global balance of the credit must be published to all participants after a given time (CREDIT_PUBLISHING_INTERVAL) to ensure transparency and equity.	The credit wallets are held in the blockchain and visible to all AU. However, for privacy reasons clearing for individual sessions are aggregated in the wallet manager and only pushed to BC in defined intervals.
priv_13	The credit balance of a participant must be accessible for the respective participant at any time.	Each AU has real-time access to its own wallet via the wallet manager and can access aggregated wallet data of all AU. The live view

ID	Description	Addressed by PE and BC
		on the wallet is created by adding the public balance and cached transactions.
priv_14	The credit balances must be consistent and immutable for any colluding minority in the system, including the platform operator.	Because the wallets are stored in the blockchain it is not possible to cheat on transactions. The zero-sum property of the token must be preserved. Additionally, we are researching NIZK methods to prevent the controller from biasing results.
priv_15	The executed flight prioritization should be transparent. *This is due to the fact that the final sequence is public anyway (checked with NMF and can be observed on runway). The individual AU inputs, however, are not revealed and zero-knowledge proof techniques will be used to show that they were correct.*	We use two mechanism to enable transparency. On the one hand, all major steps in the process will be logged in the blockchain, still keeping the privacy by only committing to sensitive data. This enables everyone to track progress and check essential properties. In the case of a dispute it is even possible to redo certain steps and verify correctness. Additionally, we are researching methods to support real-time verifiability, generation of NIZK for all steps involving private input thus enabling public verifiability (the latter part is low TRL research and will not be integrated in the final prototype).
priv_16	The final executed flight prioritization should be accessible for all AUs.	The final sequence is public and revealed to everyone for vetoing anyway. It is also recorded in the blockchain.
priv_17	The history of flight prioritization should be stored in a consistent and immutable form.	All relevant steps and results are stored to the blockchain.

Table 1: Mapping of security and privacy requirements to PE and BC components.

Additionally, to the core security and privacy requirements of the SlotMachine platform, the architecture also fulfils the specific requirements defined for the components itself. In particular, the PE requirements pe_1 to pe_17 were addressed with the design and a more detailed evaluation will be given in WP5. Also, the requirements for the MPC nodes have been taken into account (mpc_1 to mpc_13) and will be assessed in the validation phase of the project. Finally, because our implementation has been carefully benchmarked and built in a modular way, it also contributes to SM requirements regarding easy deployment, portability, and deployment as well as overall performance.

3 Components Specification

In this section we present the main component defined in the architecture of the privacy engine and describe important interfaces. Because we leverage a distributed architecture and components are implemented as micro services, the interaction between the components is important and in the focus of this specification. The micro service architecture naturally supports portability and gives us the required freedom in deployment and good scalability.

3.1 Privacy Engine Controller

The PE controller is the central management component for the PE. It manages all communication with MPC nodes and provides an easy-to-use interface with essential API endpoints to support the heuristic optimizer. A detailed specification of the interface with all data objects and options is given in *D2.2*. However, for the sake of completeness we quickly recap the most important features of the interface to the outside world. Basically, the controller offers a REST API that consumes and produces JSON-formatted data with the following end points:

- `GET /status:` This call is intended as a simple way to check the state of the privacy engine and if all MPC nodes are connected and alive.
- `GET /nodes:` Simple interface to list all configured MPC nodes.
- `PUT /sessionClear:` Input: a new weight-map (quadratic matrix) to be installed, given as a *plain-text* list of lists of integer weights
- `PUT /sessionSecret`: With this call a new weight-map is installed which basically means a new session initialisation. The weights are given as weight map in encoded form to the MPC nodes. If not successful, the lists could be of unequal length, the map does not cover all MPC nodes, the input is not a list of lists of integer values or the MPC nodes could not be reached.
- `PUT /computeFitnessClear:` This is the clear text interface to trigger the fitness computation. It takes a list of indices to pick from an installed weight-map and returns a list of integers that are the sum of the values selected from the currently installed weight-map, i.e., it returns a vector fitness values. For privacy reasons this function should be used with caution and we generally recommend using computePopulationOrder instead. However, we keep the endpoint for development and testing purposes and to integrate with different kind of heuristic optimizers. The call returns an error if there is no weight-map installed or any of the MPC nodes has returned an error, if the input is not a list of integer indices or if the MPC nodes have returned inconsistent results.
- `PUT /computePopulationOrder:` This is the main endpoint used during optimization and provides the best level of privacy in combinations with heuristic optimizers. As input, it takes a list of lists of indices to pick from an installed weight-map and outputs a `MaxOrderedResponse` object if successful. In case of error, there could be no weight map installed or the list lengths are inconsistent or any of the MPC nodes has returned an error. Also, if the input is not a list of lists of indices and the MPC nodes have returned inconsistent results, an error is raised.

- `PUT /computeClearing`: With this call the clearing is triggered. To calculate the credit balance for a chosen flight, sequence the optimal/selected solution input sent as index list. Based on the inputs and the used market mechanism a transaction matrix is computed which defines the individual transactions between flights for compensation in form of a `ClearingResponse` object. Currently implementation for SM1 and SM3 are foreseen to be supported. The usage of SM2 within the given architecture and problem modelling is under research and will eventually be integrated. On error there could be a weight-map missing or any of the MPC nodes has returned an error, alternatively the input is not a list of indices or the MPC nodes have returned inconsistent results.

- `MaxOrderedResponse`: This data object contains a privacy preserving response of fitness solutions. To not leak fitness values in clear text only an ordered version of solutions in a population of possible configurations is held additionally to the maximum fitness corresponding to the solution. In particular, the `maximum` is the clear-text maximum fitness value encountered and `order` contains the indices of the input configurations, ordered from highest to lowest fitness value.

- `ClearingResponse`: This object defines the response of a clearing computation in form of a matrix where the necessary transactions to be conducted between flights are defined. This data enables the platform to update the balances of the respective wallets accordingly.

3.2 Multiparty Computation Services

A multiparty computation service basically implements a MPC node. In a MPC protocol multiple MPC nodes jointly run a multiparty protocol to evaluate certain function on given inputs. Typically, multiparty protocols provide basic functionality and are specified to execute basic operations (gates) like addition or multiplication on integer values. To compute more complex tasks the function has to be typically decomposed into the basic gates and executed step by step (circuit). The MPC service in SlotMachine is based on secret sharing protocols and supports arithmetic gates on integer data types. This is the basis for our implementation and used to do the processing. The functions supported have been designed specifically to the project and are fixed, i.e., the circuit is static and public, which helps to achieve transparency and fairness. However, it will be easy to extend the system or add additional circuits in future version, but still we do not want the circuits to be loaded dynamically to prevent from hard to detect modifications.

MPC nodes operate on a simple TCP-based plain-text protocol. The first line of input determines the operation to be performed:

- `'PING'`: PING returns the string `PING` followed by a newline character

- `'0'`: SHUTDOWN shut down the node

- `'1'`: NEW SESSION the next line of input contains a whitespace-separated list of numbers to be used as a weight-map for subsequent computations returns the string `OK` followed by a newline character

- `'2'`: COMPUTE FITNESS the next line of input contains a whitespace-separated list of indices to be used with a previously supplied weight-map to compute a plain-text fitness value; returns the equivalent to the following Python expression followed by a newline character:

`sum([map[lin*width+ind for lin, ind in enumerate(indices)]])` (`width` being the length of the weight-map divided by length of the index vector)

- `'3'`: COMPUTE POPULATION ORDER each remaining line of input contains a list of indices as above, but the sums are to be computed without revealing them; the list of sums is then to be sorted, revealing only the original index of the list of indices from which the respective sum was computed;
  returns as a single whitespace-separated list the highest sum value encountered and the ordered list of indices, followed by a newline character

- `'4'`: COMPUTE CLEARING takes a (final) sequence as input and calculates the corresponding clearing as transaction map, which shows how many credits have to be exchanged between flights.

3.3 Encoding service

The detailed specification of the encoding service is given in *D2.2*, therefore we restrict the discussion here on the basic functionality and usage issues to consider. The idea of the encoding service is to establish a very simple and user-friendly way to encode data. Moreover, with the encapsulation of the encoding in a sub-component of the PE it is possible to change inner workings and protocols in the PE, i.e., specifically exchange the MPC protocols in use, without any changes in the software client run at the AU. Because we rely on secret sharing based MPC protocols in SM, the currently supported encoding scheme is Shamir secret sharing. The AU weight maps given in plain text json format can be directly sent to the ES and is returned in a form understandable for the PE controller and ultimately for the MPC nodes. Thus, AU sending input to the system require following steps.

- A json weight map is generated which contains one weight for each slot for a dedicated flight.
- After sending the weight map to the encoding service a version is retuned with individual weights encoded in parts (called secret shares), one for each MPC node.
- All parts destinated for a certain MPC node must then additionally be encrypted under the public key of the node.

These are the step required to privately load the input to the MPC nodes preventing any party in the system to read data in clear. However, if the encoding service in the PE is used, an additional step is required. Because the encoding service would learn the plaintext values during encoding, this would undercut the privacy requirement for the weight map. Therefore, the values must be blinded before encoding and unblinded later. This can be done by adding a random value to each weight and subtracting them from the received sharing. Thus, with this simple trick it is even possible to use the encoding service in the PE without any security implications. The main difference to the previous case is that the encryption to the value for the MPC nodes must be done locally in the application.

3.4 Blockchain Components

Most of the development in the Tendermint ecosystem builds around the Cosmos SDK and most of the code is written in Go. This is also planned for our continued development on a blockchain application, which should be responsible for two tasks: storing the credit balances for every AU and storing hashes of additional information for being able to show the proper functioning of the system in case of doubt.

The management of the credits is a priori a bit problematic, because AUs shouldn't be able to learn too much about the preferences of the other AUs. On the other hand, they should be able to gain some

insight, and to be able to see what is going on, as a basic level of transparency facilitates the trust in the system and between the AUs. Our concept is to manage the latest credit balances in a protected setup beside the blockchain by the controller, as this information is too sensitive and can reveal too much about the bidding behavior of every participant. In a given interval, the credit balances are then written to the blockchain. The interval contains several optimization rounds, which prohibits that the preferences of an AU can be directly deduced. An AU can request its own current credit balance by the controller and the credit balances, as they were disclosed the last time by the controller, of all AUs by a Tendermint node.

By this design, the system provides only a medium transparency, and because the trust from the AUs to the controller is considered to be limited, they might not be convinced in its proper operation. Therefore, there will be put a procedure in place, where AUs can decide together to disclose all input data to the optimization and control that the systems behavior has been correct in the recent past. A more sophisticated approach would involve ZK-proofs and is described in the next section. As the effort of implementing this is considered to be relatively high, it is not likely that this will be part of the demonstrator.

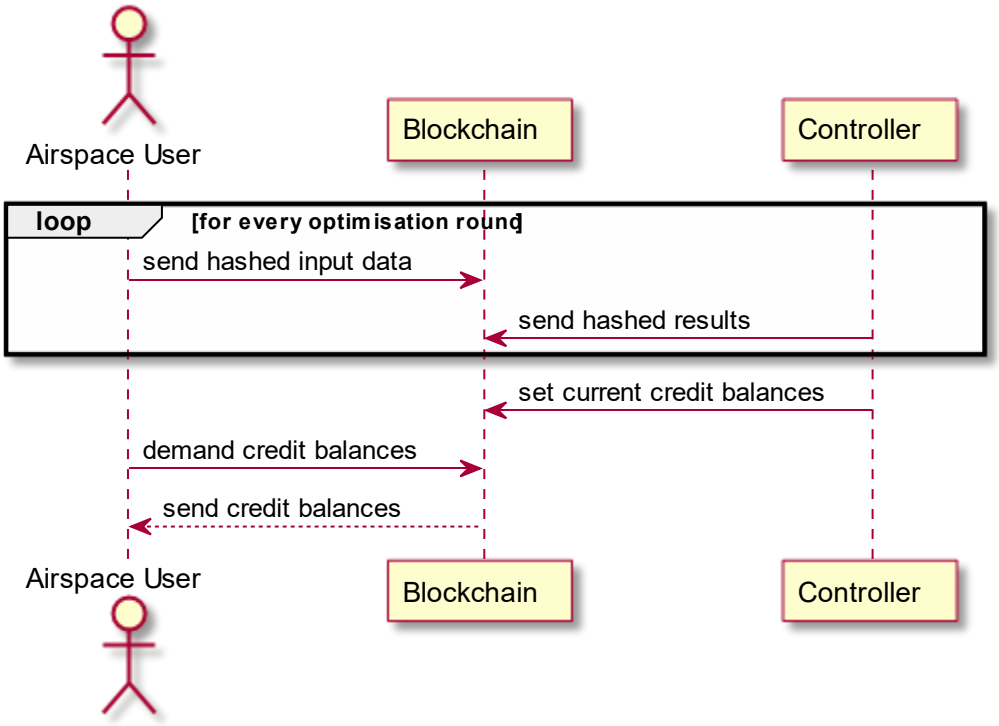

Figure 6: Basic role of blockchain component as envisaged in SlotMachine (no final)

4 Design Rationale

In this chapter we discuss positive and negative results from our research activities which built the basis and design rationale for our current approach. Parts of the work presented here, especially in MPC benchmarking and integration, were conducted in a joint effort with WP4 but are reported here for a coherent presentation of the topic.

4.1 Discrete Optimization in MPC

In this section we present our findings on fully MPC based implementation of the optimization task. The modelling approach developed as well as the expected problem sizes based on the requirements led to interesting research tasks which we tried to answer in front of the final design of the architecture. In particular we were investigating the feasibility for full-fledged optimization in MPC to get a better understanding of the problem complexity and achievable performance.

Additionally, to the optimization process, it is also relevant to do the final clearing in the privacy engine. A first analysis of clearing for *SM1* and *SM2* showed that doing the clearing in a fully privacy preserving form is feasible. This is due to the fact, that the clearing has only to be done once, after the preferred solutions are selected, and that the operations can efficiently be implemented in MPC. Therefore, we do not expect any specific difficulties in MPC based clearing calculation and only focused on possible solutions for privacy preserving optimization.

4.1.1 The Slot Assignment Problem

To understand the basic functionality required for the privacy engine we analysed the basic market mechanism introduced in *D2.3*. From a modelling point of view, the variants *SM1* and *SM3* work on the basis of a weight map which defines preferences per flight and slot. The basic modelling approach is best shown in Figure 7.

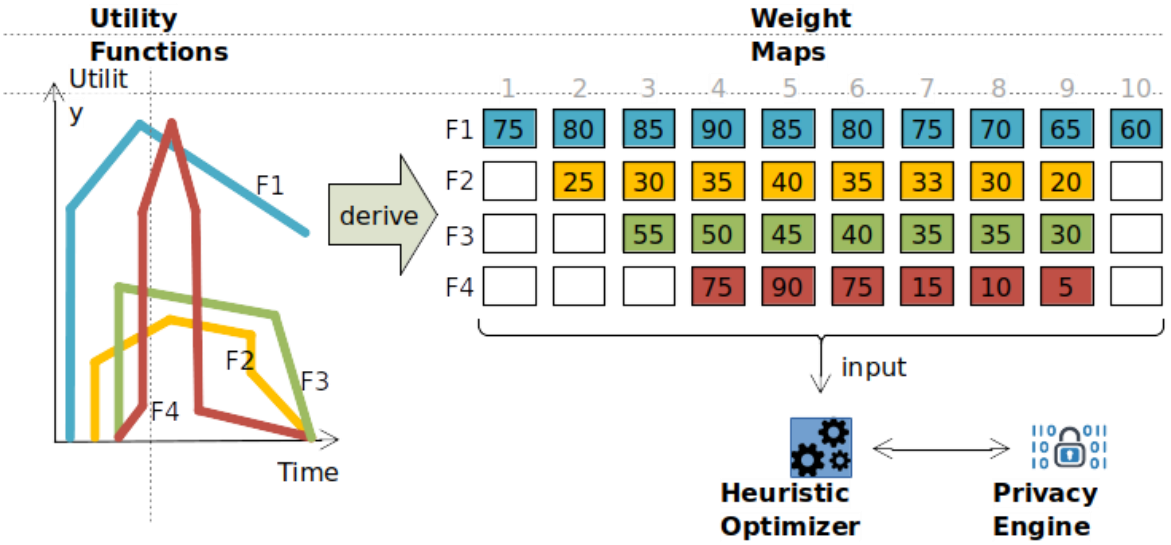

Figure 7: Overview of slot allocation problem as modelled in SlotMachine.

Only *SM2* varies from this approach and it is an open research question, how *SM2* could be modelled and optimized in the best way based on MPC. In the remainder of this part, we focus on our results achieved in benchmarking MPC based discrete optimization algorithms.

Looking carefully at the optimization problem it turns out that it basically resembles a so-called assignment problem. With the proposed objective function as sum of weights it is a linear sum assignment problem (LSAP) in particular. Luckily, the LSAP is a rather old and well understood problem where also many efficient solution strategies have been presented in the past decades. A good overview on the problem can be found in [7]. However, the challenge in SlotMachine is to do privacy preserving optimization and to find the optimum while keeping the weight matrix protected, i.e., encrypted. We therefore studied the possibilities to do discrete optimization in MPC and measured performance for different solution strategies, which seemed most promising for a MPC setting.

4.1.2 Solution strategies

A problem instance of LSAP is described by a weight matrix W, where each $w_{i,j}$ is the cost of matching vertex i of the first set (a flight in our case) and vertex j of the second set (a slot in our case). The goal of the optimization is to find a complete assignment of flights to slots which is of minimal cost according to a defined objective function, which is essential the sum of weights. Formally, let X be a Boolean matrix where $x_{ij} = 1$ iff row i is assigned to column j. Then the optimal assignment has cost

$$\min \sum_i \sum_j w_{ij}\, x_{ij}$$

$s.t.$ each row is assignment to at most one column, and each column to at most one row. In our analysis the matrix W was assumed to be quadratic, however, it can be easily generalized to a rectangular problem as shown later. If the matrix has less rows (flights) than columns (sots), then more slots are available than flights and vice versa. The latter is currently not considered in SlotMachine, because we always start from a feasible solution, the current flight plan.

A large number of algorithms has been developed for the LSAP. A good overview of solution strategies is given [7]. They range from primal-dual combinatorial algorithms, to simplex-like methods, cost operation algorithms, forest algorithms, and relaxation approaches. The worst-case complexity of the best sequential algorithms for the LSAP is $O(n^3)$, where n is the size of the problem. There is a number of survey papers and books on algorithms. Some of the most comprehensive books dealing with the topics are [8]–[10]. Additionally, interesting survey papers and tutorial like introductions are presented in [7], [11]–[13].

For SlotMachine we selected one representative for each important class of algorithms and implemented a MPC version of it to measure the practical performance which can be achieved. The selected algorithms are the

- Hungarian algorithm (aka Munkres), one of the most important candidates for the primal-dual strategy,
- Simplex based solution strategy, where we leveraged linear programming to convert the problem into network flow formulation,
- Auction algorithm, an algorithm working in the dual domain of "shadow prices".

4.1.3 Balanced vs. Unbalanced

In general, we distinguish between two types of the LSA problem, depending on the number of objects (flights) which have to be assigned to tasks (slots), i.e., the number of flights and slots in the case of SlotMachine. If the matching in a LSA is $1:1$ - every flight matches with exactly one slot - it is called balanced, and the corresponding weight matrix is quadratic with size n. It means that both parts of the bipartite graph have the same number of vertices when treating the problem as matching in bipartite graphs.

In the unbalanced assignment problem, the number of vertices is different for each side and the larger part of the bipartite graph has n vertices and the smaller part has $r < n$ vertices. In that case, either not every object can be matched to a task or not every task is occupied. Although we mostly work with balanced version in our proof-of-concept we can also cope with unbalanced situations, which are typically cases where not all slots are occupied by flights. The case of more flights than slots is not relevant for our treatment, because we always start from a feasible solution. Fortunately, most of the algorithms tested can be directly generalized to unbalanced problem solving, they can even benefit from its reduced search space.

However, even if the solver only works for balanced problems, there are methods to convert an unbalanced solution to a balanced one. The straightforward method is to add $n - r$ new vertices to the smaller part and connect them to the larger part using edges of cost 0. In our case this would mean to add $n - r$ new dummy flights with n zero weights for the slots, which corresponds to $n(n - r)$ new edges in the matching graph. Furthermore, there exists the even more efficient doubling technique [14] which requires at most $n + r$ edges to be added. The main problem with this doubling technique is that there is no speed gain when $r \ll n$.

4.1.4 MPC Aspects

Generally speaking, the algorithms presented so far are not MPC-friendly. By their nature, they are mostly sequential with very little potential for vectorized operations. One such vectorizable operation

is testing for zero. Even though this is a costly procedure in MPC that involves random number generation and comparisons, it can easily be done for a whole array in parallel, because testing one element does not involve any other elements of the same array. Also, the result can be cached, is only invalidated if the value itself changes, and can easily be recomputed on demand.

With most other operations, however, this is not possible. Take for example the minimum of a collection of elements. Finding it involves in the order of log n comparisons that have to be performed in sequence. Any change of the collection over which the minimum was computed could possibly change the minimum, so caching it is not viable. (When an element is added or changed, a single comparison is sufficient to recompute the minimum, but when an element is removed, the minimum has to be recomputed from scratch.)

To get tolerable performance we must trade-off between privacy and speed and inevitably leak some indirect information, e.g, branches been taken. However, the final assignment will be public and is known to be optimal, which also means some leakage. If that is not enough, in Aly and Cleemput have shown how to efficiently implement graph algorithms that, like ours, reveal branching information, yet do not leak information by just obliviously permuting the original data [15].

Another problem is that every algorithm that uses some form of ϵ-scaling needs to use floating-point numbers. This is not just a question of numerical stability. If the underlying numerical representation is not precise enough, ϵ-scaling may terminate with a solution that is not optimal or may not even terminate at all. In his survey, Bertsekas [12] proposes multiplying every element of the $n * n$ matrix by $(n + 1)$ and use only integer values (down to 1) for ϵ but notes that this may in practice lead to integer overflow because prices can then be somewhere in the order of $n^2 \max_{(i,j)\in A}|a_{ij}|$.

4.1.5 Hungarian Method

Our first implementation is based on the Hungarian algorithm, also known as the Munkres or Kuhn-Munkres algorithm [16]–[18]. It is one of the first polynomial-time algorithms published to solve the balanced assignment problem but can also be easily adapted to the unbalanced case. It is a global algorithm improving a matching along augmenting paths and therefore alternating paths between unmatched vertices. Its run-time complexity when using Fibonacci heaps is $O(mn + n^2 \log n)$, where m is a number of edges in the corresponding bipartite graph. This is currently the fastest run-time of a strongly polynomial algorithm for this problem. If the weights are integers, and all weights are at most C (where $C > 1$ is some integer), then the problem can be solved in $O(m\sqrt{n}\log(n \cdot C))$ weakly-polynomial time in a method called weight scaling [19].

size	time (sec)	min (count)	is_zero (count)
10	0.8	646	314
20	6	6544	3246
30	28	27767	13842
40	53	51473	25688
50	113	99512	49697
60	247	202795	101320
70	448	355287	177547
80	711	525472	262626
90	1156	781422	390585
96	1487	1013148	506432

10	0.9	646	708
20	7	6544	8404
30	31	27767	38871
40	61	51473	72873
50	128	99512	151452
60	276	202795	331680
70	517	355287	600210
80	803	525472	889063
90	1313	781422	1350726
96	1860	1013148	1719199

Table 2: Benchmark results for different problems sizes with a normal MPC based Munkres and a manually optimized version.

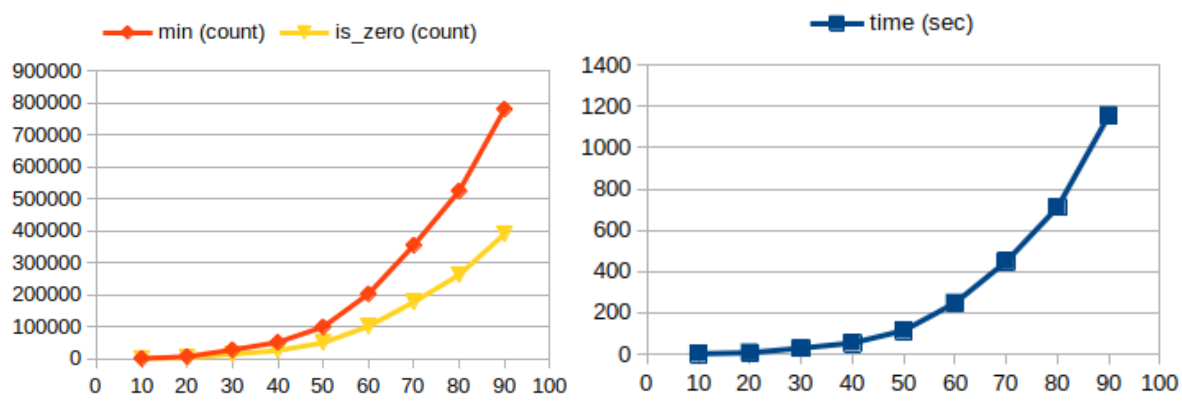

Figure 8: Visualisation of benchmarking results over various problem sizes.

The performance achieved in our MPC implementations are shown in Table 2. The table shows the duration of an optimization run in seconds depending on the problem size. It also contains information about the amount of costly MPC operations needed (minimum finding and zero testing). The upper part of the table represents results from an optimized implementation compared to the textbook version below. The results show a 3x improvement with our manual optimization, which is significant but still too slow for our application in SlotMachine, even in ideal conditions without network latency. The results are also visualized in Figure 8.

4.1.6 Linear Programming

The assignment problem is a special case of the transportation problem, which is a special case of the minimum cost flow problem, which in turn is a special case of a linear program[1]. Therefore, it is natural to ask what the practical performance of an MPC solver based on the simplex algorithm is.

[1] Wikipedia:HungarianAlgorithm

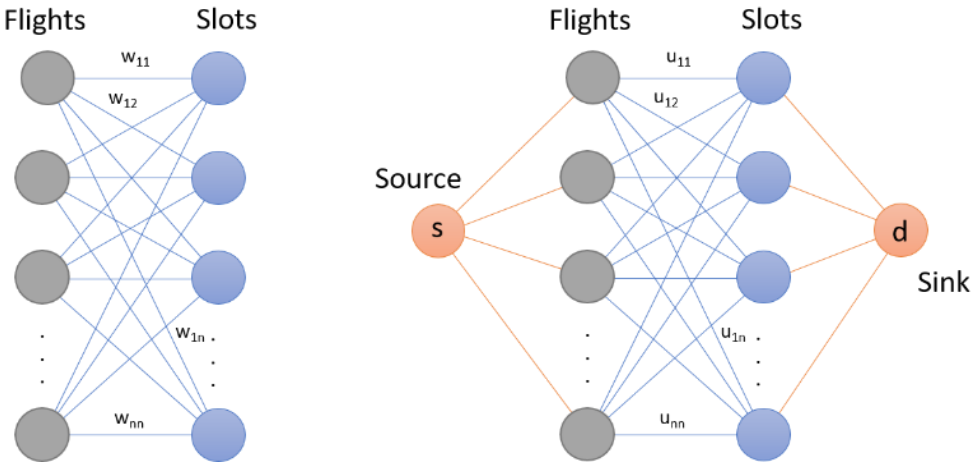

Figure 9: Modelling approaches as LP problem. Left it is shown as weight minimization in bipartite graph and right as flow maximization problem with virtual source (s) and sink (d) nodes.

The assignment problem can be solved by presenting it as a linear program. This may be rather counter intuitive because one would more likely expect the formulation as an integer program because of the binary nature of a match. However, because the constraint matrix of the fractional LP is an unimodular matrix, the optimal solution will always take integer values with better run time compared to directly solving integer programming formulation.

For convenience we will present the maximization problem, but this can be converted easily to a minimization. For the problem formulation we start with a bipartite graph as depicted in Figure 9 (left part). Each edge (i, j), where i is in A and j is in T, has a weight w_{ij}. For each edge (i, j) we have a variable x_{ij}. The variable is 1 if the edge is contained in the matching and 0 otherwise, so we set the domain constraints $0 \leq x_{ij} \leq 1$ for $i, j \in A, T$, The total weight of the matching is $\sum_{(i,j)\in A\times T} w_{ij}\, x_{ij}$, which is the objective function we have to maximize for a perfect matching. To guarantee that the variables indeed represent a perfect matching, we add constraints saying that each vertex is adjacent to exactly one edge in the matching. This assures the 1:1 mapping between flights and slots, i.e, $\sum_{j\in T} x_{ij} = 1$ for $i \in A$, $\sum_{i\in A} x_{ij} = 1$ for $j \in T$.

All in all, we have the following LP if we formulate it as maximum-weight matching problem.

$$\text{maximize} \sum_{(i,j)\in A\times T} w_{ij}\, x_{ij}$$

$$\text{subject to} \sum_{j\in T} x_{ij} = 1 \text{ for } i \in A, \quad \sum_{i\in A} x_{ij} = 1 \text{ for } j \in T$$

$$0 \leq x_{ij} \leq 1 \text{ for } i, j \in A, T,$$

This is an integer linear program we can solve without the integrity constraints as discussed above. Additionally, to the maximum weight matching formulation we can represent the problem also as a maximum flow problem in a slightly modified graph as shown in Figure 9 (right part). By introducing a source vertex s and a sink t and treating the weights as upper capacity bounds, we can define a similar LP also delivering the same maximum matching. In the flow formulation the objective function maximizes the single commodity network flow under flow conservation and upper bounded edges. We

implemented the flow variant and report benchmark results in Table 3. From the results it can be seen, that this approach is even much slower than Munkres and not feasible for use in SlotMachine. However, is the most flexible approach when it comes to modelling and interesting in its own right.

size	time (sec)	min (count)	is_zero (count)	iterations
20	59	0	21398	42
30	240	0	70074	66
96	10840	0	na	239

Table 3: Benchmark results form MPC based simplex solver for max flow LP.

4.1.7 Auction Algorithm

Additionally, to the methods presented, two variants of auction algorithms have been benchmarked. Auction based algorithms were identified as interesting candidate for MPC implementation, because they often lead to good practical performance, although worst case performance is the same as for Hungarian ($O(N^3)$), and have potential for MPC based customization and optimizations. Auction algorithms were introduced in 1979 and have since then evolved as a valuable tool in network optimization [20]. For a detailed presentation, we refer to the survey paper [12] and the textbooks [8], [9].

We quickly recap the basic ideas of the method as presented in [20]: The auction algorithm is based on an economic equilibrium problem that turns out to be equivalent to the assignment problem. Consider our problem of matching n flights with n slots through a market mechanism, viewing each flight as an economic agent acting in his own best interest. There is also a benefit a_{ij} for matching flight i with slot j. Suppose that slot j has a price p_j and that the flight who receives the slot must pay the price p_j. Then, the (net) value of slot j for flight i is $a_{ij} - p_j$ and each flight i would logically want to be assigned to a slot j_i with maximal value, that is, with

$$a_{ij_i} - p_{j_i} = \max_{j=1,\dots,n} \{a_{ij} - p_j\}.$$

Flight i is considered happy if this condition holds and we will say that an assignment and a set of prices are at equilibrium when all persons are happy. The equilibrium assignment offers maximum total benefit (and thus solves the assignment problem), while the corresponding set of prices solves an associated dual optimization problem. This is a consequence of the celebrated duality theorem of linear programming. Additionally, in [21] it was shown that the original auction algorithm and the Goldberg&Kennedy algorithm [22] – another efficient solver - are equivalent.

We implemented two auction algorithms in MPC and performed many experiments to understand their performance potential and to compare them. The experiment shows that also in the case of MPC the auction algorithm performs and scales better in practice than the other algorithms, which are also harder to implement and still have the same worst-case complexity.

Implementation 1: MPC version of SciPy LSAP solver. The first implementation was based on the implementation available in the open source SciPy module[2]. The core algorithm was implemented as MPC version and many manual optimizations were tested and compared. The best performance values achieve are summarized in Table 4. From there it can be seen, that even in ideal conditions with no network latency at all it is not possible to fulfil the performance requirements for the full problem size, however, the results are promising and may be relevant for other applications.

n → Latency (ms) ↓	10	20	30	40
0	3,26	15,32	47,30	100,97
2	5,33	25,93	81,56	177,45
4	7,38	36,92	117,45	257,88
6	9,50	47,96	152,30	337,19
8	11,61	59,25	189,38	417,71
10	13,77	70,48	225,72	499,72

Table 4: Performance results in seconds (duration of optimization run) for given problem size n and increasing network latency in ms.

Implementation 2: MPC version of ACM-ICPC solver. Our first version of a privacy preserving auction algorithm was based on the freely available implementation from the implementation of the Stanford ACM-ICPC teams[3]. The best achieved performance after careful manual optimization is shown in Table 5. Compared to implementation 1, this version performs even better (about 30-40%), however, no further optimization potential could be identified, and the achieved performance seems almost optimal in the given setting, because main parallelization and caching strategies have been tried and compared. Therefore, also this approach does not fulfil the responsiveness necessary for SlotMachine.

n → latency (ms) ↓	10	20	30	40
0	2,42	9,87	33,58	72,81
2	3,36	14,60	52,95	118,77
4	4,36	19,46	73,01	166,47
6	5,36	24,34	93,25	213,98
8	6,49	29,41	113,93	261,73
10	7,41	34,43	134,65	310,11

Table 5: Performance of MPC version of ACM-ICPC solver in seconds for given problem size n and increasing network latency in ms.

[2] https://github.com/scipy/scipy/blob/v1.7.0/scipy/optimize/rectangular_lsap/rectangular_lsap.cpp
[3] https://github.com/jaehyunp/stanfordacm/blob/master/code/MinCostMatching.cc

4.2 MPC for Heuristic Optimization

The main results from our research activity show, that a full MPC implementation of a deterministic optimization algorithm for the expected problem size is not possible. This was already the hypothesis at the beginning of the project and has now been empirically proven. Therefore, the use of a heuristic optimizer was intensively studied in WP4 also considering the privacy aspects. The resulting architecture developed employs evolutionary algorithms and uses the MPC system as a co-processor, therefore achieving both, the required performance and response times but still achieving strong privacy for AU preferences.

The basic concept behind the combination of the heuristic optimizer with a MPC system is to still keep private input in encrypted form only but relax the computational requirements for the MPC part compared to the full MPC optimizations. In essence, the privacy engine encapsulates MPC functionality and provides an interface to compute the fitness for a given set of possible flight sequences once the AU preferences are loaded into the system. The heuristic optimizer uses this interface to find good solutions which are almost optimal without access to the weights, i.e., the sensitive input set by the AUs.

Although this approach is extremely fast and elegant it raises an issue regarding the privacy of the weights. With each query the heuristic optimizer learns something about the weights and if it is able to ask enough queries it would be able to recover the full weight map. As a simple example, for a problem size of n flights and slots, a malicious heuristic optimizer would be able to recover all n^2 individual weights after n^2 random queries with high probability by solving a linear system of equations. Thus, privacy could even be compromised by an honest but curious platform operator, which we want to protect from as a minimum requirement. A naïve solution to address this would be to limit the number of queries the heuristic optimizer can query or to add noise to the results as typically done in differential privacy tools. However, as an analysis has shown, both options can significantly degrade the quality of the solutions the optimizer finds, thus, alternative approaches were needed.

To address the leakage via fitness computations we therefore implemented additional privacy friendly versions. In one approach, we only reveal the order of the solutions but not the individual fitness except for the maximum. This solution prevents the leakage but still provides enough information for the heuristic optimizer to work properly. However, because this solution implies the implementation of a relatively resource consuming sorting algorithm in the MPC system we are also researching even more extreme variants, e.g., only revealing the best quarter of sequences. Finding the optimal trade-off for different kind of optimization algorithms is still under investigation in SlotMachine, however, we envisage to address this problem to further improve SlotMachine even beyond what is absolutely required.

4.3 Towards Public Verifiability

Additionally, to the integration of MPC we are also researching means to enhance the trustworthiness of the SlotMachine platform. In the SlotMachine prototype we will record various relevant information during the swapping process in the blockchain such that incorrect behaviour can be detected. However, for sensitive information only digest information or commitments can be stored in a publicly accessible blockchain. This makes verification more cumbersome and requires a dedicated offline procedure with all parties present to fully verify the correctness of the process. Therefore, we are also

working on an improved protocol with public verifiability, thus enabling online auditing in real-time. However, this work is early research and not part of the main SlotMachine prototype. Many challenging research questions have still to be solved to realize such a system with practical efficiency. In this report we present our current approach in this directions and intermediary results achieved. More work will be done during the last project phase and hopefully we can show the technically feasibility at all, which still requires novel ways to proof the optimality in heuristic optimization or optimization of large combinatorial problems at all.

The basic idea behind improved trustworthiness in the decentralized system is to integrate non-interactive zero-knowledge proof of knowledge (NIZK) methods to make the process verifiable, albeit being privacy preserving. This enables on-line verifiability in contrast to the demonstrator version, where digests of important steps are recorded in the blockchain to prevent stakeholders from cheating in the protocol and to enable offline verifiability in case of a dispute. However, because this is a very challenging task at very low TRL and a high technical risk, we are mainly researching protocols and only implement basic functionality in a standalone proof-of-concept not integrated with the fully fledged demonstrator developed in WP5.

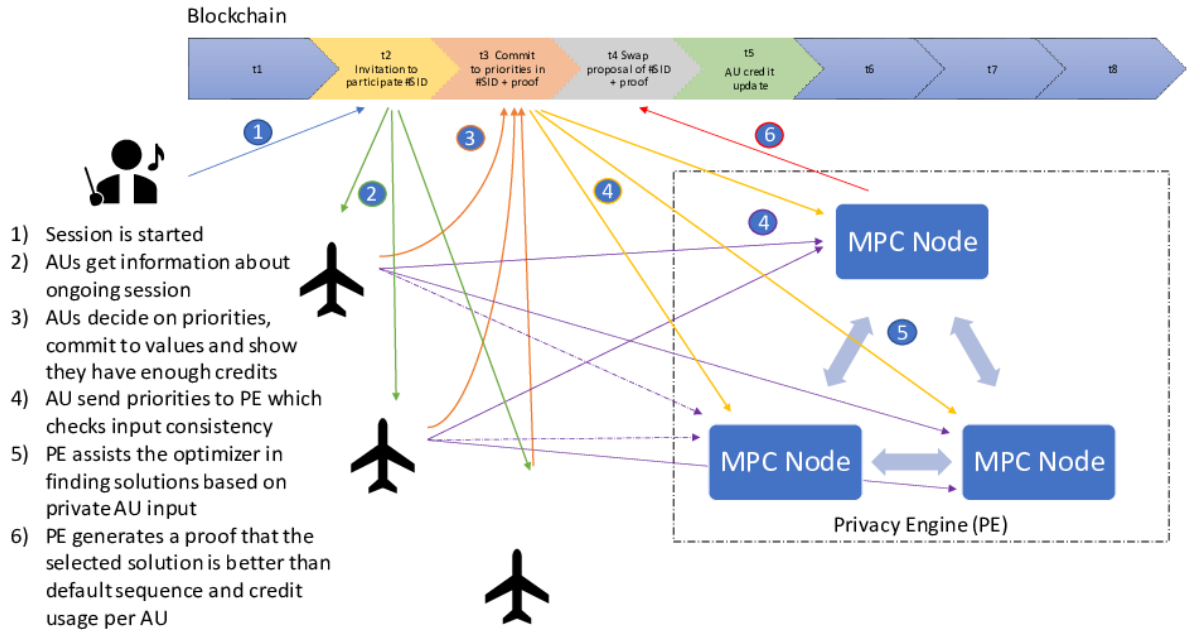

Figure 10: High level overview of data flow with MPC and blockchain for public verifiability.

In the following we present the current status of the developed architecture for a verifiable and privacy-preserving decentralized slot management platform which supports public verifiability. The envisaged operation and data flow for the combination of MPC with ZKP also leveraging blockchain is shown in Figure 10. In our scenario, we consider an optimization platform, at which AUs are registered for certain airports to participate in the swapping process. A dedicated optimization session is then started by an orchestrator —the controller in our case— and AUs are informed about opportunities to participate. The AUs can then input preferences for their flights in form of margins, priorities and optionally credits, depending on the market mechanism used. In the verifiable PoC, we are then leveraging multi-party computation to ensure confidentiality of individual inputs, blockchain for immutability of AU input and results, and zero-knowledge proofs to ensure integrity and verifiability.

To do so, the participating AUs also have to commit to the input sent to the MPC system in the blockchain and the platform will additionally attach a proof of correctness to the announced optimal sequence generated. If the proof of correctness is stored in the blockchain it can be verified by anybody given access to the blockchain, in our case also the AUs. Although the basic data flow is somehow straight forward, many challenges must be solved to realize such a system with practical performance. At the time of writing the framework was developed and first basic functionality was demonstrated, however, there are still many research questions ahead which have to be solved, e.g., it is unclear how the final proof can be efficiently computed and how credit handling can be included. Nevertheless, in this report we show the progress made and present the approach in more detail.

4.3.1 Security Objectives

In the following, we review most important security and privacy objectives to be targeted with the next generation architecture. Please note, this is a rather high-level treatment for a possible next generation version developed in a follow-up project and beyond the scope for the currently ongoing developments in WP5 according to requirements defined in *D2.1*. It basically extends certain properties of the current demonstrator. In the end, the new properties of the system should lead to increased willingness to participate. More precisely, the requirements are as follows.

Confidentiality. Confidentiality of the AUs' input is of utmost importance through all phases of the auction. In particular, the margins and weights do not only need to be protected from unauthorized access through competitors, but also from the platform provider, which serves as a global optimizer. This is because of the risk of this central entity colluding with certain producers, thereby fully undermining the price finding mechanisms.

Integrity. Besides the requirement of correctness in the case of exclusively semi-honest entities it is necessary that the integrity of an optimization result can also be guaranteed in the case of a malicious operator of the platform. This even needs to hold in the case that the provider is colluding with other entities in the system, i.e., certain MPC node operators or AU, to ensure that no party can manipulate the outcome of the auction in their own interest.

Availability. While this is often not considered in the design of cryptographic protocols, it turned out to be of high importance to our partners. On the one hand, users demand assurance that they will not miss opportunities. On the other hand, related to integrity, producers also need to be guaranteed that they cannot be excluded from an optimization run; that is, whenever an AU places preference for a flight, it shall also be guaranteed that these preferences were indeed considered.

Anonymity and Pseudonymity. In addition to confidentiality of input, AUs may also wish to even hide the information whether or not they placed priorities for a given auction, as this might already reveal sensitive information about the current internal status. Depending on the specific business model of the marketplace, this requirement needs to be balanced against the platform provider interest, which sometimes need the information.

Rank Fairness. Another important aspect for the clients was fairness in terms of fair conditions. This can also be interpreted as an open and transparent way of optimizing in a transparent and unbiased way. In particular, the objective function used to rank different swapping solutions must be publicly know.

Transparency. Finally, transparency requires that all participants in the system are able to trace progress and activities on a high level. The users may also be interested in historical data to further optimize their planning activities. However, all this functionality needs to be achieved without compromising any of the previous goals, especially those related to confidentiality and privacy.

These requirements are also in line with previous work regarding security and privacy of auctions found in the literature and discussed in D3.1; i.e., bid privacy, posterior privacy, bid binding, public verifiable correctness, financial fairness, non-Interactivity.

4.3.2 Optimization and Clearing Mechanism

Different market mechanisms have been proposed in *D2.3* and various optimization approaches evaluated in *D4.1*. In summary many different combinations of market solutions and optimization strategies and for practical application more than a single one has to be supported by a generic framework. In that sense, it is essential to have a very flexible mechanism for ranking solutions when optimizing and price clearing options.

From a MPC point of view the proposed mechanisms do not introduce a specific challenge and can be implemented efficiently, however, regarding privacy and public verifiability many challenges arise which are not fully solved. Because the clearing is also not fully specified at the time of writing, we leave the further treatment for the final project phase but consider already known prerequisites already in the theoretical treatment of the framework.

4.3.3 Framework

The proposed framework is designed to address the security objectives defined in *D2.1*, especially bringing together typically contradicting goals of privacy and verifiability in a single solution. The framework is designed as a decentralized architecture which incorporates edge computing capabilities managed by users connected to a central cloud infrastructure where the platform is hosted. The system design is following data minimization principles and data is not uploaded if it can be processed locally. Additionally, using secure multiparty computation the platform itself is operated in a way that the provider does not learn sensitive data, thereby minimizing the necessary trust assumptions as even certain malicious behaviour would be detected. This is achieved by making all steps in the data flow verifiable. To do so, publicly verifiable zero-knowledge proofs of knowledge are generated for all computations. To trace all interactions and proofs we use a distributed ledger which serves as a trust anchor and immutable append-only data base.

The system enables end-to-end verifiable computations in a flexible manner. On one hand, it enables the edge to participate and pre-process data, therefore minimizing the amount of data which need to be uploaded to the platform. On the other hand, it additionally achieves security for the processing in the cloud combining inputs from different parties. The framework supports typical application patterns which are found in many other application scenarios but are specifically relevant for SlotMachine.

4.3.4 Data Flow

In the following we detail the data flow in our platform. To ease understanding, Figure 11 provides a high-level overview, where we omit setup steps for the sake of clarity.

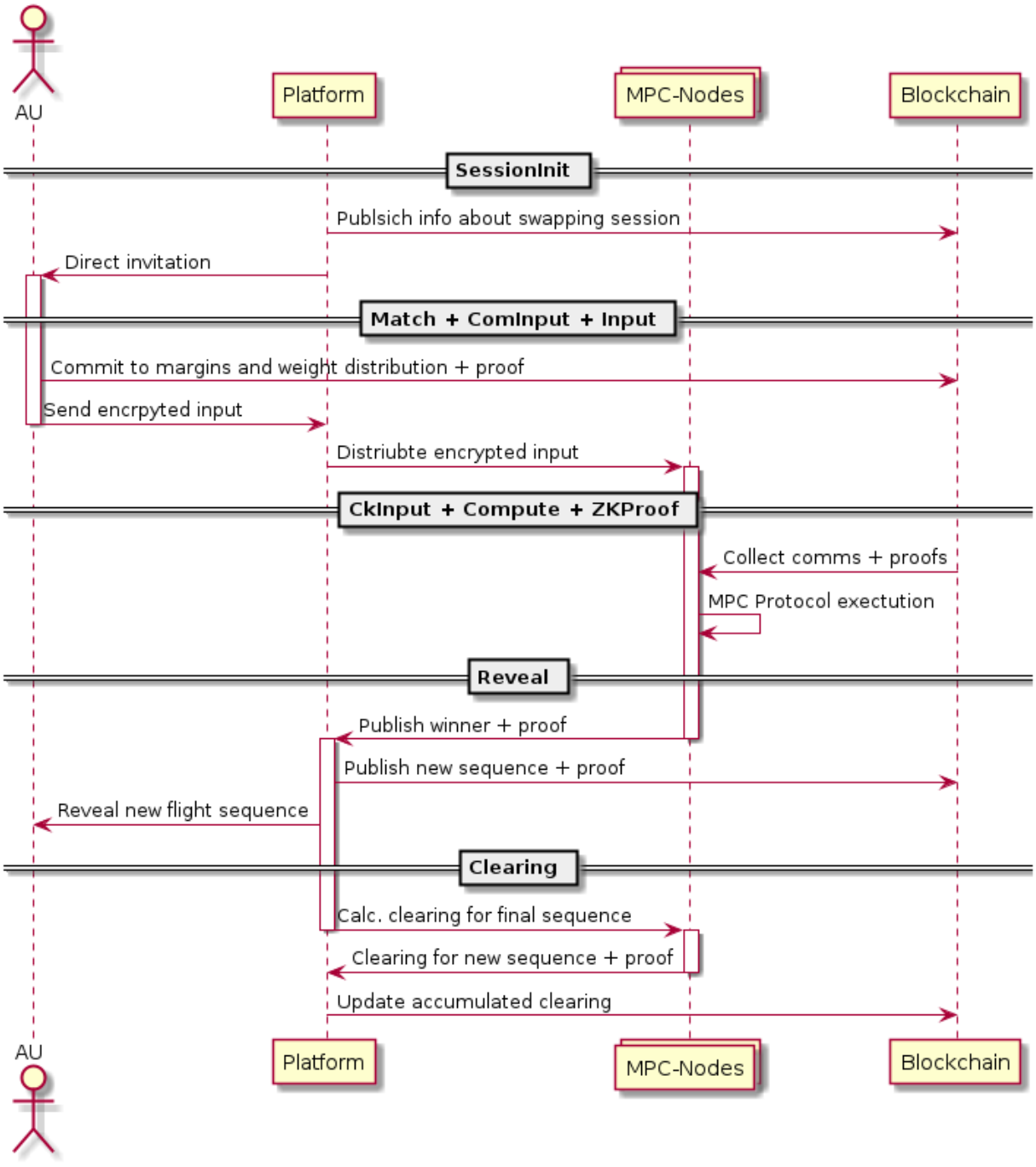

Figure 11: Session overview for an extended slot swapping session with public verifiability as currently under investigation in SlotMachine.

4.3.4.1 Setup phase.

The following setup steps are necessary to operate the system.

- On the one hand, **ZKSetup** is used to generate the common reference string (CRS) needed for the NIZKs. On giving as input the security parameter and a circuit, this algorithm outputs the CRS which is assumed to be an implicit input to all further algorithms and parties. It is important to note that the system shall be designed in a way that this step is only needed once and does not need to be invoked again if different ranking mechanisms are used, as they are all supported by the specifically designed circuit with built-in flexibility. In practice this setup algorithm can be run in dedicated setup ceremony, including, e.g., secure hardware elements or dedicated MPC-based ceremonies.

- On the other hand, **RegUser** is a protocol which is run by the user and the platform to register with the platform. It is used to generate necessary identities and credentials to authenticate the user and set up the necessary permissions on the ledger.

4.3.4.2 Operative phase.

After the setup is complete the following steps are conducted in the protocol for a particular auction.

- **SessionInit**. An orchestrator sends a swapping session request with relevant parameters to all parties and stores them in the blockchain.

- **Match**. Based on the session information received, the AUs check if they are eligible for participation and if they have an interest in prioritizing. If an AU decides to participate, they calculate margins, priorities, and credits they want to set. Additionally, the data is pre-processed to generate the weight map which is ultimately used within the MPC, i.e., the privacy engine. If an AU does not want to participate, the local process is aborted.

- **ComInput**. Cryptographic commitments for the inputs together with the computed weight map are generated. Additionally, a NIZK showing the correctness of the weight map computation based on the original margins and priorities is generated. Finally, a proof showing that the weight map fulfils policies for weight distribution according to session parameters could be also computed if required. The commitments together with corresponding proofs are then stored in the blockchain.

- **Input**. In this step the AU sends their margin data together pre-processed weight distribution to the MPC system in a secret-shared fashion.

- **CkInput**. The MPC system retrieves the corresponding commitments and proofs from the blockchain and verifies them in the encrypted domain. This is done by recomputing the commitments on the shares (for margins and weight maps) at each node and comparing the reconstructed commitments with the plaintext ones. Additionally, each node verifies the proof for the local pre-processing at each AU individually. If either of the checks fails, the system complains about the AU.

- **Compute**. The MPC system runs the optimization task and calculates the rankings for different swapping scenarios based on the inputs it is holding. This can be by either running a full deterministic optimization, if there is enough time to do the computation on encrypted data, or by assisting the heuristic optimizer if faster execution times are needed, as is the case in SlotMachine. The final overall score for the (almost) optimal solution may be published if needed or sent to the heuristic optimizer in the platform.

- **ZKProof**. The MPC system generates a NIZK for the (almost) optimal solution, proving that it is the best ranked result according to the predefined ranking function and the optimization mechanism. It does so by each node computing the proof on its share of the result.

- **Reveal**. To reveal the result in a verifiable form, the optimal swap is revealed together with a commitment on the optimal score and a NIZK showing the score is correct. Furthermore, an additional proof showing the selected swap is also (almost) optimal should be produced. This is currently the main challenge and under development. The data is recorded in the blockchain and finalizes a particular auction.

- **Clearing**. Optionally, the clearing together with NIZK shall be produced and sent to the platform where it is reconstructed. Also, the privacy preserving clearing process is not fully understood and under research at the time of writing. However, we see a need for a credential management component in the platform which manages per session clearings and only pushes aggregated clearings to the blockchain to preserve privacy of the AU inputs.

There are many variations possible in practice, but this will only result in subtle changes, e.g., if the optimal score could be made public.

4.3.5 Protocols

Different protocols have been used, extended, and integrated to achieve all desired properties for our framework. At the core we combine multiparty computation with zero-knowledge proofs of knowledge to achieve confidentiality and public verifiability that the same time. Regarding MPC we do not rely on any specific protocol but only require a method which is based on secret sharing. However, because we aim at public verifiability the correctness of the computation is going to hold even if all nodes are corrupt. Therefore, depending on the individual assumptions made for the MPC deployment, it can be sufficient to rely on passively secure protocols for input privacy.

To achieve verifiability, the system is based on adaptive zk-SNARKs as introduced in [23]. Working with commitments to track different steps in the process is essential to guarantee privacy of sensitive data. However, the protocol is not guaranteeing any authenticity which is essential to track the flow from end to end. Therefore, we leverage ideas from ADSNARK [24] and use signatures on the commitments to assure the authenticity of the data right from the source. In our use cases both can be used, standard signatures but also group signatures, if a certain degree of anonymity is still required, e.g., if it should not be visible which department of a larger organization is managing certain flights. Additionally, by simply signing the commitments we achieve more flexibility because the commitments support batch operation on data vectors.

Finally, an important goal is to reduce the number of times the CRS setup procedures have to be executed. Ideally, it has only to be done once when initializing the platform and can then be reused for all subsequent auctions.

For our use case a hybrid approach is aspired. On the one hand we intend to use the idea of subroutines, i.e., predefined subroutines which are defined at setup time but can be connected during proof generation by means of intermediate commitments, to establish the required circuit. This concept is very flexible with only little overhead, i.e., the additional commitments increase the proof size and verification time for each subroutine defined. To enable even more freedom in the configuration of ranking algorithms we research possibilities to integrate universal circuits based on

ideas from MIRAGE [25]. Altogether, we intend to use circuitry which comprises both, static elements and freely re-configurable components to get the best of both worlds. To that end, this approach is somehow similar to partially reconfigurable hardware.

4.3.6 Security

In the following we will informally discuss the achieved security goals; a more formal treatment is planned as future work. Moreover, we also give some design rationale and explain several architectural decisions.

Confidentiality. The privacy of sensitive information is protected in our framework by two main primitives. On the one hand, MPC is used to compute the winner of the auction in a privacy preserving way, as sensitive data is protected by the input privacy provided by MPC.

On the other hand, to enable transparency we are recording inputs at different stages in the blockchain. To achieve confidentiality there we use commitments which are also hiding input. Given that sensitive inputs are never handled in cleartext in the system we achieve strong cryptographic protection, which also results in the discussed properties of bid privacy and posterior privacy.

In our marketplace, we even apply a decentralized input pre-processing. In the current approach the AUs can directly pre-process the margins into a weight distribution for each flight. Through this approach they are more flexible in fine-tuning the weights within a given policy defined for correct input. Alternatively, the PE would have to generate the weight distribution from margins given by AUs, which would cause more load on the PE and gives less flexibility to AUs.

Integrity and correctness. In essence, the basic idea of the framework is to preserve the integrity and authenticity of data in the system and to prove the correctness of each computation in between. We use commitments on each step of the process to track progress in the system and assure authenticity by signing them during upload into the blockchain. The blockchain itself serves as immutable public database or bulletin board. Due to the hiding property of the commitments, privacy of input data is still preserved, while guaranteeing that producers are bound to their bids. The framework is based on the extractable commitments presented in [23]. In the original work these commitments were used with a dedicated key to distinguish between input from different parties, but this key is produced in the initial setup phase and must be distributed to parties, which opens up many attack vectors in practical implementations. We rely on locally generated private keys, which never leave the local area and are registered with the platform or the blockchain. Alternatively, the use of group signatures would allow for even more flexibility in the management of edge components without sacrificing the security.

After the optimization session is initiated and all AUs recorded the input in the blockchain, the MPC based optimization is started. Contrary to a normal MPC model where inputs are sent to the system and the result is sent back to the parties, we employ an augmented view. The input is comprised of the private bid, the private matching score as well as the data also recorded on the blockchain, i.e., the commitments on initial machine parameters and the matching score accompanied by a NIZK, thereby guaranteeing confidentiality while still binding bidders to all input values. Finally, the MPC system not only outputs the winning bid, but also a NIZK proving its optimality, thereby guaranteeing the integrity of the final result.

It is worth noting that the performed local pre-processing of weight distribution introduces another problem with the integrity of data: It has to be assured that the weights were computed correctly and fulfil the policy for correct input. Therefore, the AUs are required to generate a proof on the weights set when they are sent to the MPC system. We do so by forcing the AUs to commit not only to the margins but also to the weight distribution for the flights and additionally to generate a NIZK showing the correctness of the weight distribution. All data is then uploaded to the blockchain before sending the input to the MPC system.

It is important to note, with this approach we are extending the security model of the platform beyond that of MPC. As the correctness of the computation becomes publicly verifiable by means of NIZKs, the integrity of the computation can even be assured if all MPC nodes maliciously deviate from the protocol specification. Even more, in our setting malicious behaviour can be attributed to the right stakeholder, i.e., it is not possible to blame the platform for malicious input from bidders or vice versa. This is achieved by letting the MPC system check all inputs for consistency with the information in the blockchain before it computes a result. Only if all inputs are consistent with the stored commitments and the matching score is computed correctly, the MPC system will incorporate the bid in the auction, and only then it will be able to compute a proof for the winning bid.

As a result, the full data flow is accompanied with NIZKs and every participant can verify the correctness of the optimization session from end-to-end. Even if privacy is compromised by an adversary which compromises enough MPC nodes to recover the inputs, he will still not be able to influence the outcome of the optimization process. Depending on the use case, it would also be possible to leverage very efficient MPC protocols from a user point of view, because the correctness property of the auction could be directly verified.

Availability. The availability of the system is assured by the blockchain component which provides the properties to serve as robust and immutable public append-only log. Depending on the deployment of the MPC system, also robustness properties such as fairness or guaranteed output delivery are achieved. Additionally, as the system is non-interactive, client-side computations cannot be interrupted or blocked by individual participants, resulting in a highly available decentralized architecture. Although the platform server is currently needed to run auctions it would also be possible to remove this single point of failure, but this scenario is not relevant for SlotMachine.

Anonymity and Pseudonymity. For the given use case it is not desired to build a completely open and permissionless infrastructure. The AUs (clients) in this system are part of an ecosystem which requires some level of assurance to operate and have no direct need for anonymity. However, some AUs might not want to leak whether or not they participated in a given session. This leakage can be easily avoided by always participating in respective sessions with ∞ weights.

Fairness. In our prototype, the swap session information is public and could thus be also put into the blockchain, which also serves as a broadcast channel in this step. Therefore, all participants are reliably informed about new possibilities as well as the detailed parameters and criteria for the optimization algorithm used.

Transparency. By logging every step to the blockchain in a privacy-preserving way and also proving that all computations are correct, we achieve public verifiability. Every user of the system will thus be able to verify all auctions based on the public data stored in the blockchain without compromising the privacy of individual inputs, thereby achieving the requirement of transparency.

4.4 Credit Balances with Zero-Knowledge Proofs

It would be desirable, that every party can check the correct behaviour of the system. For the credit balances this means that an AU can only spend credits it possesses and furthermore results in a correct change of the balances. It is relatively hard to respect this property in an implementation without reducing the desired level of privacy. An approach to achieve this is by adding ZK-proofs and adapting the interactions between the participants slightly.

Additional to sending the input via the controller to the Privacy Engine, an AU also commits to the input on the blockchain. The PE then calculates a ZK-proof in addition to the clearing and sends it also to the blockchain. The clearing is forwarded to the controller, which updates all the balances privately and sends each AU its current balance with an ZK-proof. With this procedure, the AUs are able to verify that the credit balances are calculated correctly.

This extension to the system won't be fully implemented as part of this project as the expected effort would be too high. Nevertheless, the approach will be taken in consideration while describing and building the components. This way, the system will be designed in an extensible manner and the ZK-proofs can be added on top at a later time. It could also be worth investigating if the wallet management can be lifted to the PE, so that the balances can even be kept secret from the controller.

5 Summary and Conclusion

In this report we present the architecture developed for the privacy engine and blockchain which are essential components in SlotMachine to achieve security and privacy goals. The privacy engine enables developer-friendly access to multiparty computation, a method to compute on encrypted data. Additionally, the blockchain component is used to run a permissioned distributed ledger with a dedicated blockchain application (smart contract). The blockchain serves as an immutable public storage and computation system which is accessible to all stakeholders.

The document provides implementation details about the functioning and inner workings of the different (sub-)systems as well as descriptions of important interfaces for both, external and internal ones. Additionally, we give some design rationale to support our decisions and show positive and negative research results achieved on the way. Finally, we present the status of ongoing research activities for improved functionality. In summary, we could verify our research hypothesis and confirm our decision for the use of a dedicated heuristic optimizer, because standalone MPC based implementation are without reach for SlotMachine. We also identified future research topics regarding transparency and verifiability which are currently not mature enough to be integrated with the SlotMachine prototype but could lead the way for future developments for the privacy engine beyond the current project.

In a next step, privacy engine and blockchain components will be implemented according to this specification and integrated with the overall platform as planned in *D2.2*. In parallel, more research on the clearing process will be done to enable seamless integration for token support with the current architecture. Furthermore, research on open topics (e.g., public verifiability, SM2 integration) will continue merely on a theoretical basis and optionally with exploratory standalone proof-of-concept implementations.

6 References

[1] SlotMachine Consortium, „D2.2 System Design Document“, SlotMachine Report, 2021.
[2] SlotMachine Consortium, „D2.1 Requirements Specification“, SlotMachine Report, 2021.
[3] SlotMachine Consortium, „D3.1 Report on the State-of-the-Art of Relevant Concepts“, SlotMachine Report, 2021.
[4] SlotMachine Consortium, „D2.3 Business Concepts“, SlotMachine Report, 2021.
[5] SlotMachine Consortium, „D4.2 Specification of Evolutionary Algorithms“, SlotMachine Report, 2021.
[6] SlotMachine Consortium, „D4.1 Report on the State of the Art of Relevant Concepts“, SlotMachine Report, 2021.
[7] R. E. Burkard und E. Çela, „Linear Assignment Problems and Extensions“, in *Handbook of Combinatorial Optimization*, 1999. doi: 10.1007/978-1-4757-3023-4_2.
[8] T. B. Boffey und D. P. Bertsekas, „Linear Network Optimization: Algorithms and Codes.“, *J. Oper. Res. Soc.*, Bd. 45, Nr. 4, 1994, doi: 10.2307/2584223.
[9] D. P. Bertsekas, „Network optimization : continuous and discrete models“, 1998.
[10] R. Burkard, M. Dell'Amico, und S. Martello, *Assignment Problems*. Society for Industrial and Applied Mathematics, 2012. doi: 10.1137/1.9781611972238.
[11] M. Akgül, „The Linear Assignment Problem“, in *Combinatorial Optimization*, Berlin, Heidelberg, 1992, S. 85–122.
[12] D. P. Bertsekas, „Auction algorithms for network flow problems: A tutorial introduction“, *Comput. Optim. Appl.*, Bd. 1, Nr. 1, S. 7–66, 1992, doi: 10.1007/BF00247653.
[13] M. Dell'Amico und P. Toth, „Algorithms and codes for dense assignment problems: the state of the art“, *Discrete Appl. Math.*, Bd. 100, Nr. 1, S. 17–48, 2000, doi: https://doi.org/10.1016/S0166-218X(99)00172-9.
[14] L. Ramshaw, R. E. Tarjan, R. E. Tarjan Princeton, und H. P. Labs, „On Minimum-Cost Assignments in Unbalanced Bipartite Graphs“, *HP Lab.*, 2012.
[15] A. Aly und S. Cleemput, „An Improved Protocol for Securely Solving the Shortest Path Problem and its Application to Combinatorial Auctions“. 2017.
[16] H. W. Kuhn, „The Hungarian method for the assignment problem“, *Nav. Res. Logist. Q.*, Bd. 2, Nr. 1–2, 1955, doi: 10.1002/nav.3800020109.
[17] H. W. Kuhn, „Variants of the hungarian method for assignment problems“, *Nav. Res. Logist. Q.*, Bd. 3, Nr. 4, 1956, doi: 10.1002/nav.3800030404.
[18] J. Munkres, „Algorithms for the Assignment and Transportation Problems“, *J. Soc. Ind. Appl. Math.*, Bd. 5, Nr. 1, 1957, doi: 10.1137/0105003.
[19] H. N. Gabow und R. E. Tarjan, „Faster Scaling Algorithms for Network Problems“, *SIAM J. Comput.*, Bd. 18, Nr. 5, S. 1013–1036, 1989, doi: 10.1137/0218069.
[20] D. P. Bertsekas, „Auction Algorithms“, 2009.
[21] C. A. Alfaro, S. L. Perez, C. E. Valencia, und M. C. Vargas, „The assignment problem revisited“, *Optim. Lett. 2021*, S. 1–18, Aug. 2021, doi: 10.1007/S11590-021-01791-4.
[22] A. V. Goldberg und R. Kennedy, „An efficient cost scaling algorithm for the assignment problem“, *Math. Program.*, Bd. 71, S. 153–177, 1995.
[23] M. Veeningen, „Pinocchio-Based Adaptive zk-SNARKs and Secure/Correct Adaptive Function Evaluation“, Springer, Cham, 2017, S. 21–39. doi: 10.1007/978-3-319-57339-7_2.

[24] M. Backes, M. Barbosa, D. Fiore, und R. M. Reischuk, „ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data“, in *2015 IEEE Symposium on Security and Privacy*, Mai 2015, S. 271–286. doi: 10.1109/SP.2015.24.

[25] A. Kosba, D. Papadopoulos, C. Papamanthou, und D. Song, *MIRAGE: Succinct Arguments for Randomized Algorithms with Applications to Universal zk-SNARKs*. 2020.