

D3.3 PrivacyEngine Software Component

Deliverable ID:	D3.3
Dissemination Level:	PU
Project Acronym:	SlotMachine
Grant:	890456
Call:	H2020-SESAR-2019-2
Topic:	SESAR-ER4-27-2019 Future ATM Architecture
Consortium Coordinator:	Frequentis AG
Edition date:	23th November 2022
Edition:	01.01.00
Template Edition:	02.00.05

Authoring & Approval

Authors of the document

Name / Beneficiary	Position / Title	Date
Thomas Lorüenser / AIT	WP3 Leader	30.9.2022
Florian Wohner / AIT	Engineer	16.9.2022
Stephan Krenn / AIT	Senior Scientist	16.9.2022
Roman Karl / AIT	Engineer	16.9.2022

Reviewers internal to the project

Name / Beneficiary	Position / Title	Date
Eduard Gringinger / FRQ	Project Manager	13.10.2022
Christoph Fabianek /FRQ	Technical Manager	13.10.2022

Approved for submission to the SJU By - Representatives of all beneficiaries involved in the project

Name / Beneficiary	Position / Title	Date
Christoph Fabianek / FRQ	Technical Manager	23.9.2022
Eduard Gringinger / FRQ	Project Manager	23.9.2022
Marie Carré / SWISS	WP 5 Leader	23.9.2022
Nadine Pilon / EUROCONTROL	WP2 Co-Leader	23.9.2022
Christoph Schuetz / JKU	WP2, 3 Leader	23.9.2022
Lorünser Thomas / AIT	WP4 Leader	23.9.2022

Document History

Edition	Date	Status	Name / Beneficiary	Justification
00.00.01	5.9.2022	Draft for approval	All AIT	
01.00.00	30.9.2022	Final version	Thomas Lorünser / AIT	Final corrections
01.01.00	23.11.2022	Revised version	Thomas Lorünser / AIT	Review comments

Copyright Statement © 2022 – SlotMachine Consortium. All rights reserved. Licensed to SESAR3 Joint Undertaking under conditions.

SlotMachine

A PRIVACY-PRESERVING MARKETPLACE FOR SLOT MANAGEMENT

This Deliverable 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 document we report the software implementation results for essential components developed in work package 3. In particular, proof-of-concept implementations for the SlotMachine privacy engine and blockchain components were developed, 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 provides general information the software implementations developed in the project and delivered for later integration in the full SlotMachine Platform. After a quick overview of the specified software architecture and relevant interfaces we give all information to access, build, and deploy the components. We also update the specification developed one year ago where needed and summarize the achieved security and scalability with the current version. Finally, we conclude the work and highlight necessary actions to reach higher technology readiness levels.

Note: The opinions expressed herein reflect the author's view only. Under no circumstances shall the SESAR Joint Undertaking be responsible for any use that may be made of the information contained herein.

Table of Contents

Abstract	3
1 Introduction.....	6
1.1 Purpose of the document.....	6
1.2 Scope	6
1.3 Intended readership	6
1.4 Background	6
1.5 Structure of the document and relation to other deliverables	7
2 Software Architecture	8
2.1 Overview.....	8
2.2 Privacy Engine	8
2.3 Blockchain.....	11
3 Software Components.....	12
3.1 Privacy Engine	12
3.2 Blockchain.....	17
4 Specification Update	19
4.1 Privacy Engine	19
4.2 Blockchain.....	19
5 Security and Scalability	22
5.1 Privacy Engine	22
5.2 Blockchain.....	24
6 Conclusion and Outlook.....	26
7 References	28

List of Tables

Table 1: Overview of PE software repository.....	12
Table 2: Rank of equation system given by the set of all $n!$ possible swap permutations for given problem size n with n^2 unknown weights.	23
Table 3: Performance in seconds to compute output for given API calls to PE.....	23

List of Figures

Figure 1: Components and important interfaces of the PE.	9
Figure 2: Sequence diagram demonstrating how to use the PE in normal operation for one optimization session.	10
Figure 3: PyTest example and demo for component level testing (API tests).	14
Figure 4: Docker deployment configuration for PoC with 3 MPC nodes and one PE instance.....	15
Figure 5: Swagger UI interface description as provided by PE service.....	16

1 Introduction

1.1 Purpose of the document

In this document we present the final version of our proof-of-concept software implementation of the privacy engine (PE) and associated components as well as the blockchain (BC) solution as delivered in SlotMachine. 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. The Blockchain component (BC) is used as trust anchor, i.e., as reliable, robust and accessible append only database, and to manage delay credits/tokens as used in the SlotMachine market mechanism. The software modules have been released as open-source software and the documents is intended as basic manual.

1.2 Scope

This is an accompanying document to the final software implementation for PE and BC delivered in SlotMachine. It summarizes the current state of the software implementation and how it can be used. In that sense, it describes the realization of the components specified in *D3.2* [1] as integrated into the platform specified in *D2.2* [2], including some minor updates necessary during the course of the development.

1.3 Intended readership

The document is intended as a quick introductory manual to get started with the software modules for PE and BC. It serves as a starting point for internal partners who integrate the components into the platform. Additionally, it may also be interesting for external audience who want to study and experiment with PE and BC components in its own rights, because both components are also published as open-source software.

1.4 Background

The current implementation is intended to be used in the final proof-of-concept of the SlotMachine platform. It fulfils core requirements regarding security, privacy and transparency as described in *D2.1* [3] and integrates with the overall platform specified in *D2.2*. The design was developed in close cooperation to WP4 and the team working on the optimizer, which is documented in *D4.2* [4].

The final version integrates research results also published in [5], [6], [7], and [8]. However, most recent results from [9] are not fully included, because it will require a change in the software architecture which cannot be done within the remaining time in the project and is left for future work. Nevertheless, the proof-of-concept fully satisfies the requirements and new research results also show the potential for future development in potential follow-up projects.

1.5 Structure of the document and relation to other deliverables

The document introduces and describes the final version of our proof-of-concept implementation delivered in SlotMachine. It is intended as a “get started manual”, therefore, the remainder is structured as follows.

In chapter 2 the software architecture of the two components developed in work package 3 are recapped to give a quick overview.

Chapter 3 is dedicated to the detailed explanation of the repositories containing the software implementations released as open source. The code structure is introduced, installation and usage guidance are given as well as relevant software development topics are discussed.

Because we followed an agile development approach the first version of our software specification released in D3.2 was subject to minor changes and adaptations. In chapter 4 we summarize the updates on the specification which were necessary during the final development phase.

In chapter 5, we discuss the security and scalability achieved with the current version and discuss potential improvements.

Chapter 6 provides the conclusion and discusses remaining research topics to be addressed after the project to further increase TRL level as well as security and performance.

2 Software Architecture

2.1 Overview

The basic idea behind SlotMachine is to build a trustworthy distributed platform which can be operated by stakeholders without the need for a central authority. It contains no single point of trust, especially with respect to the confidentiality of sensitive user input data, and protected data are only processed in encrypted form. Additionally, crucial steps shall be made verifiable by logging essential checkpoint data in a blockchain.

Concretely, to ensure confidentiality of flight preferences from airspace users, they are first encoded for the use in the embedded secure multi-party computation (MPC) system and are sent to the respective MPC node. During the operation of the platform the sensitive data is processed in encrypted form only and never decoded thanks to MPC protocols used. Thus, data is protected even from operators of MPC nodes by the very nature of the protocols used. All these mechanisms are encapsulated in the privacy engine component (PE), which enables easy and seamless use of complex cryptographic protocols. In that sense, the PE encapsulates all cryptographic work into modules which can be used without deeper cryptographic knowledge over a simple and intuitive application programming interface.

Nevertheless, a certain amount of leakage in the setting of SlotMachine is inevitable for the system to work, e.g., a newly computed flight sequence must eventually be published by the Network Manager. This is intrinsic to the application of privacy preserving computation techniques, where the clients always learn information they can deduce from the result of the computation and their own input. In our example, the airlines know their own preferences but also the new optimal flight sequence, which is public information. Therefore, the trade-off between the information that is needed for the platform to operate while still maintaining the confidentiality of sensitive information has been carefully studied and respected in the design of the component [9], [10].

As a general remark, we want to mention that the research focus was on the exploration of novel cryptographic methods and their practical impact. In particular we researched and developed novel methods for secure data processing and verifiable computing. Additionally, the use of secure connections between the different components is necessary and recommended for real deployments in production environments. However, we did not use secure connections in our lab deployments, because it helped to focus on the research topics and allowed for more efficient testing and evaluation of the core technologies. Nevertheless, the switch to secure connections can be achieved by setting up a public key infrastructure and enabling standard means (TLS and VPN) in the configuration settings.

2.2 Privacy Engine

The Privacy Engine (PE) encapsulates the functionality for secure computation of fitness values of candidate solutions. The PE is responsible for the management and protection of confidentiality of sensitive input submitted by the participants in encrypted form while conducting computations over the inputs using MPC. The PE provides a REST interface for the Heuristic Optimizer to invoke the PE and employs MPC nodes, accessed via TCP protocol, to conduct the computations (Figure 1).

The nodes maintain separate TCP connections with each other, operated and controlled by the underlying MPC framework. In addition, each participant locally runs an encoding service, which allows

participants to encode/encrypt inputs before sending the inputs to the MPC nodes, preventing information leakage.

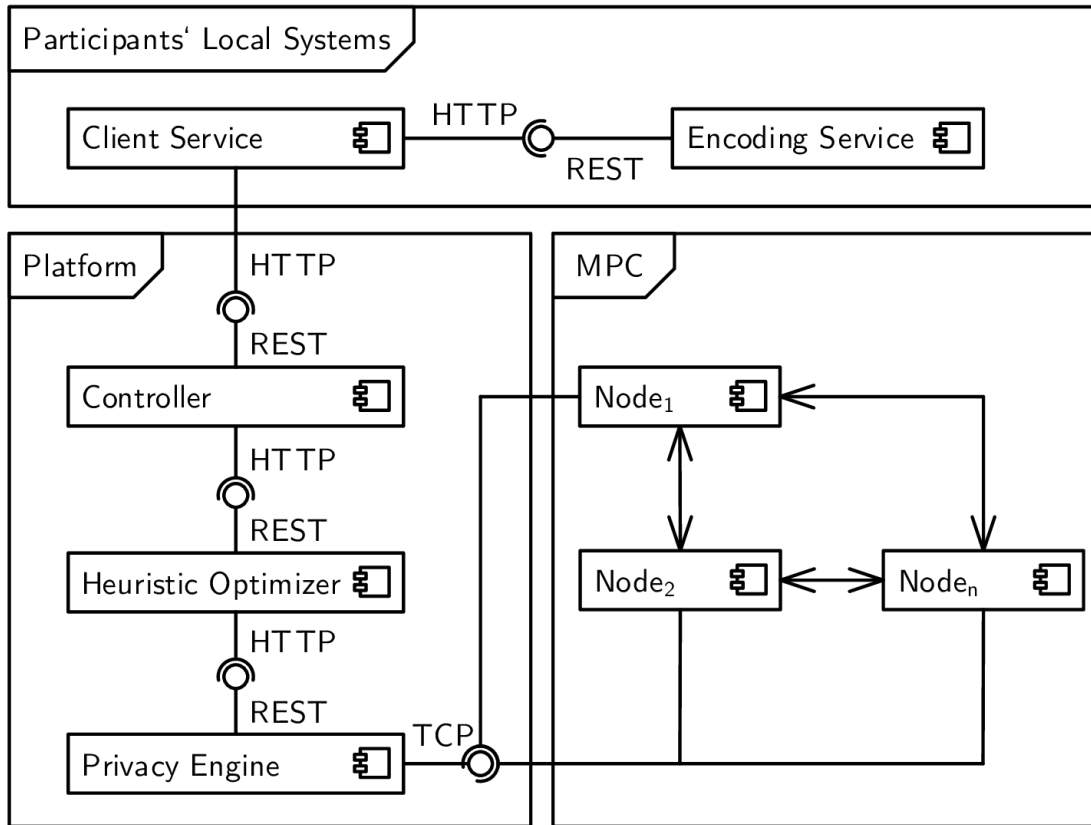


Figure 1: Components and important interfaces of the PE.

The PE enables the Heuristic Optimizer to compute the relative fitness of populations without revealing the underlying inputs, i.e., the submitted weights (utilities) for combinations of slots and flights. Thus, after the weight maps have been communicated to the PE, the optimizer can then invoke the PE to compute aggregates over the weights in a privacy-preserving way. If 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 participants.

To better understand how the PE should be used we present a typical usage pattern. The PE operates on the basis of optimization runs. An optimization run is characterized by a list of flights and a list of (time) slots as well as by the inputs (preferences) submitted by the participants. In the following, we present the privacy-preserving session for normal operation.

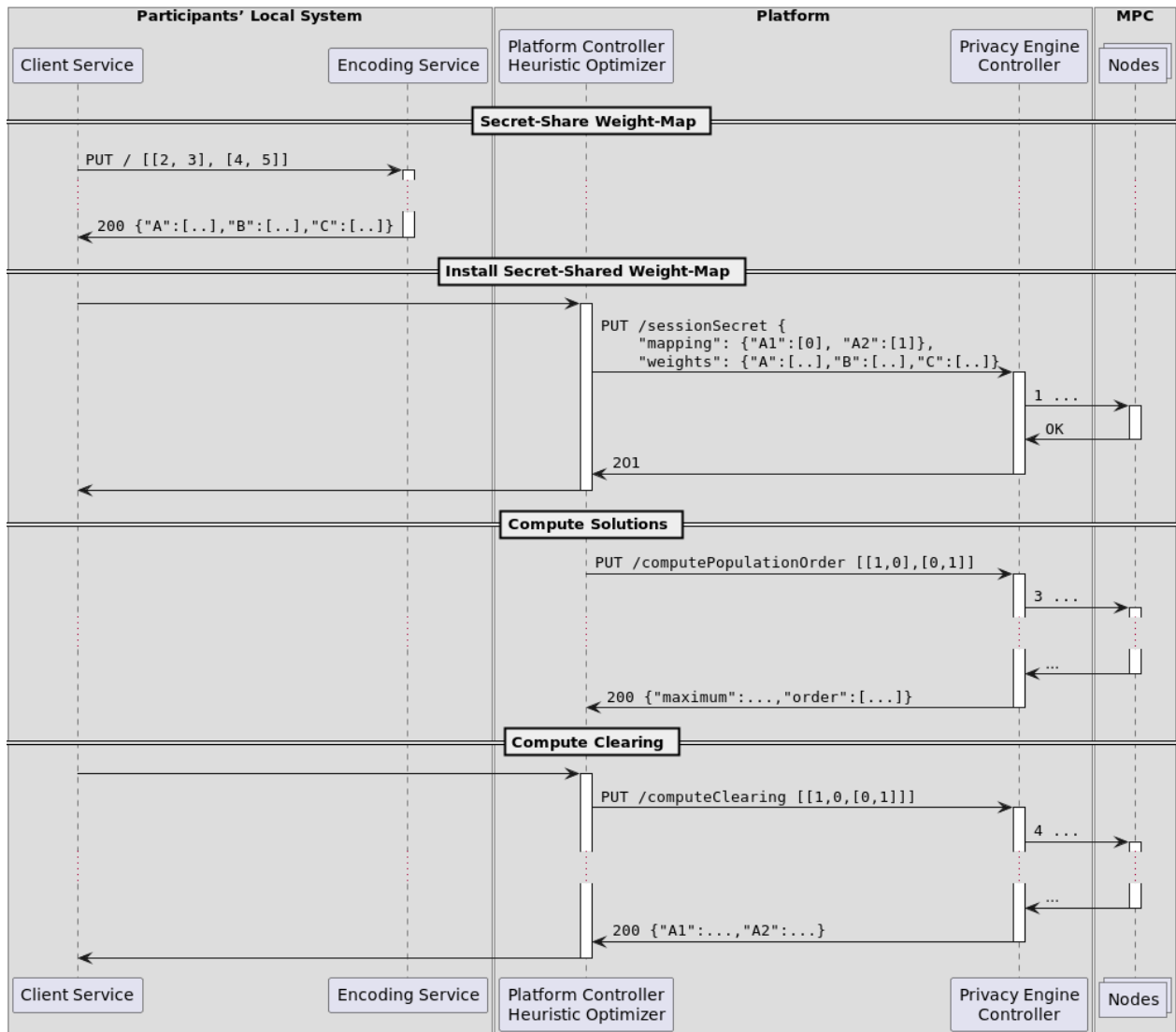


Figure 2: Sequence diagram of the call sequences supported by the Privacy Engine; interaction between Client Service, Platform Controller, and Heuristic Optimizer is described in detail in D 4.3.

The encoding service, locally run by each participant, turns a plain-text weight map into a secret-shared form that is suitable input for the MPC nodes. The different shares are encrypted for the respective MPC nodes and sent from the participants' local systems to the PE via the platform's controller component. In Figure 2 we show the essential message sequences to interact with the PE and the corresponding encoding sever. Additional platform communication is omitted for readability. The detailed message flows for the overall platform are presented in D4.3 [11] in detail.

The PE implementation builds on various Python libraries and integrates MPC using different frameworks.

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, each container (encoder, controller, MPC node) is only about 13 MB in size.

Furthermore, the overall memory footprint during execution is low since the PE does not store any data beyond optimization runs and only keeps encrypted input data during the run; no additional information is logged, and all computations are executed on demand.

2.3 Blockchain

A blockchain-based system is used as a trustworthy storage and as an archive visible for all the participants. It is used for two different tasks: as credit archive and as event protocol. The credit archive stores the balances for all the AUs. Having the latest credit balances at every time visible for other AUs would allow them too much insight in the internal values for slots of an AU. Therefore, the latest credit balance is kept private at the controller and only written at a slightly higher time interval to the blockchain. As a second part, the event protocol can be thought of as a persistent log file with a moderate log level. It is not intended as a fine-grained log that can be used to debug single applications, but as a protocol for certain events that are relevant from a systems perspective. It also provides basic means for querying the recorded events.

The trustworthy storage is based on Tendermint Core and other parts of the Tendermint ecosystem. With Tendermint there is a clear separation between the application logic and the consensus mechanism. The application logic is contained in a blockchain application, which is a similar concept as a smart contract with the difference that the code of the blockchain application is kept off-chain. The consensus mechanism ensures that blocks will stay part of the chain and cannot be replaced by blocks of a different branch as in other blockchain-based systems like Bitcoin and Ethereum. For the functioning of the overall system, it is required that more of the half of the nodes in the network keep working and are honest. The minimal number of nodes that allow for one dysfunctional or dishonest node is four, which is also our chosen network size for the proof-of-concept. In production, every AU would be encouraged to run its own node but would also have the possibility to cooperate with another AU and use its Tendermint node.

3 Software Components

3.1 Privacy Engine

3.1.1 Software Repositories

The whole project is written in Python and uses the `FastAPI` web framework with the `uvicorn` ASGI web server. MPC functionality is provided by the `MPyC` framework. `PyInstaller` is used to obtain compact binaries that can be easily bundled together with the small `Alpine Linux` Docker container, yielding container images that are only around 13 MB in size for each of the Privacy Engine components (Controller, Encoder, MPC Node). The structure of the software repository is shown in Table 1 below.

<ul style="list-style-type: none"> ▼ app <ul style="list-style-type: none"> Dockerfile main.py requirements.in requirements.txt 	<p>Controller Component</p> <p>Dockerfile to build container Implementation in a single Python file requirements.in: turned into requirements.txt with pip-compile</p>
<ul style="list-style-type: none"> ▼ enc <ul style="list-style-type: none"> Dockerfile encode.py requirements.in requirements.txt 	<p>Encoder Component</p> <p>Dockerfile to build container Implementation in a single Python file requirements.in: turned into requirements.txt with pip-compile</p>
<ul style="list-style-type: none"> ▼ mpc <ul style="list-style-type: none"> Dockerfile mpc_node.py requirements.in requirements.txt 	<p>MPC Node</p> <p>Dockerfile to build container Implementation in a single Python file requirements.in: turned into requirements.txt with pip-compile</p>
<ul style="list-style-type: none"> > venv .gitignore docker-compose.yml README.md 	<p>See below for contents of <code>docker-compose.yml</code></p>

Table 1: Overview of PE software repository.



The software is also available as open source at the AIT software repositories under the SlotMachine-Public Folder¹ The folder comprises two repositories relevant for the PE.

On the one hand, in PrivacyEngine-Public the source code for the all components relevant to operate the PE are available.

- <https://git-service.ait.ac.at/sct-slotmachine-public/privacyengine-public>

On the other hand, in the PrivacyEngine-Container repository, the pre-built containers can be found and directly downloaded and deployed.

- <https://git-service.ait.ac.at/sct-slotmachine-public/privacyengine-container>

The software was developed in Python according to known best practices. To avoid certain pitfalls, we also rely on additional typing information using type hints and variable annotation which improves code quality. Additionally, we implemented test routines for all modules to enable continuous testing and integration and provide some additional test scripts to run component level integration tests as show in Figure 3.

Complete PrivacyEngine is built with `docker-compose build`, started with `docker-compose up -force-recreate`, then tests are run with the help of `pytest`.

¹ <https://git-service.ait.ac.at/sct-slotmachine-public>

```

import random

import requests

from json import JSONDecoder
from operator import itemgetter
from random import shuffle, randrange

MAT_SIZE = 16
POP_SIZE = 16

decoder = JSONDecoder()
app = "http://127.0.0.1:80"
enc = "http://127.0.0.1:8081"
matrix = [[randrange(start=0, stop=2**16) for _ in range(MAT_SIZE)] for _ in range(MAT_SIZE)]
mapping = {"One": [0, 1, 8, 9], "Two": [2, 3, 10, 11], "Three": [4, 5, 12, 13], "Four": [6, 7, 14, 15]}

def test_status():
    response = requests.get(app + "/status")
    assert response.status_code == 200

def test_init():
    response = requests.put(enc + "/", json=[[2, 3], [4, 5]])
    assert response.status_code == 200
    json = decoder.decode(response.content.decode())
    json = {"mapping": {"One": [0], "Two": [1]}, "weights": json}
    response = requests.put(app + "/sessionSecret", json=json)
    assert response.status_code == 201

```

```

(venv) florian@l309sqt01 ~/C/privacyengine-container (main)> pytest test.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: /home/florian/Code/privacyengine-container
plugins: anyio-3.6.1
collected 11 items

test.py ..... [100%]

===== 11 passed in 3.03s =====

```

Figure 3: PyTest example and demo for component level testing (API tests).

3.1.2 Installation and Deployment

In the base directory, the `docker-compose.yml` file contains everything needed to build and run the complete Privacy Engine. Issuing the command `docker-compose build` will download the necessary base containers and libraries, and build the containers.

Then, the command `docker-compose up --force-recreate` will start the fully pre-configured Privacy Engine and make the Controller interface available at localhost, port 80.

For our deployment we chose to use a 3-party MPC system.

Each component of the Privacy Engine is built as a separate container: one Encoding Service (`enc`), one Controller (`pe`), and three MPC Nodes (`mpc1`, `mpc2`, `mpc3`). The Encoding Service has to know each MPC Node's address, which is accomplished by setting an environment variable called `PEERS` containing a dictionary mapping nodes to their addresses. The MPC Nodes each have to know their own port and the other Nodes' addresses, which is accomplished by the `HP1`, `HP2`, `HP3` environment variables (where addresses with ports signify other nodes, and a port without address signifies the

node itself). To facilitate integration with other SlotMachine components, the Node addresses are symbolic and will be resolved at runtime to actual network addresses by the container orchestration framework.

```

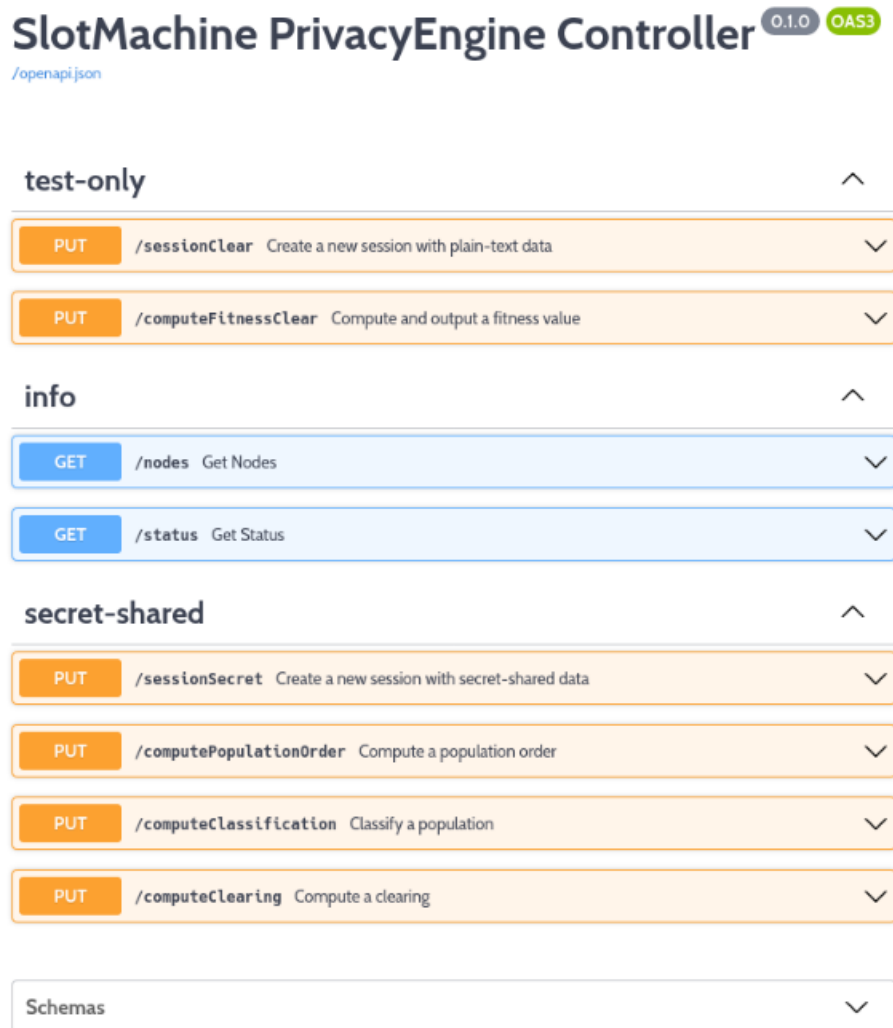
1  services:
2    pe:
3      image: git-service.ait.ac.at:8010/sct-slotmachine-public/privacyengine-container/pe-controller
4      ports:
5        - "127.0.0.1:80:8000"
6      environment:
7        - PEERS={"A":"http://mpc1:1234", "B":"http://mpc2:1236", "C":"http://mpc3:1238"}
8      networks:
9        - mpc_net
10   mpc1:
11     image: git-service.ait.ac.at:8010/sct-slotmachine-public/privacyengine-container/mpc_node
12     environment:
13       - HP1=:1234
14       - HP2=mpc2:1236
15       - HP3=mpc3:1238
16     expose:
17       - "1234"
18       - "1235"
19     networks:
20       - mpc_net
21   mpc2:
22     image: git-service.ait.ac.at:8010/sct-slotmachine-public/privacyengine-container/mpc_node
23     environment:
24       - HP1=mpc1:1234
25       - HP2=:1236
26       - HP3=mpc3:1238
27     expose:
28       - "1236"
29       - "1237"
30     networks:
31       - mpc_net
32   mpc3:
33     image: git-service.ait.ac.at:8010/sct-slotmachine-public/privacyengine-container/mpc_node
34     environment:
35       - HP1=mpc1:1234
36       - HP2=mpc2:1236
37       - HP3=:1238
38     expose:
39       - "1238"
40       - "1239"
41     networks:
42       - mpc_net
43   enc:
44     image: git-service.ait.ac.at:8010/sct-slotmachine-public/privacyengine-container/pe-enc
45     ports:
46       - "127.0.0.1:8081:8000"
47   networks:
48     mpc_net:
49       ipam:
50         driver: default
51         config:
52           - subnet: 192.168.0.0/24
53             gateway: 192.168.0.12
54

```

Figure 4: Docker deployment configuration for PoC with 3 MPC nodes and one PE instance.

3.1.3 Usage

The Controller includes interactive API documentation, provided by the `Swagger UI` library. It is automatically generated from the method definitions and comments in the source code by the `FastAPI` framework and accessible under `127.0.0.1/docs`, as shown in Figure 5. In addition, a static Open API JSON Schema file can be downloaded there.



SlotMachine PrivacyEngine Controller 0.1.0 OAS3
/openapi.json

test-only ^

- PUT** /sessionClear Create a new session with plain-text data v
- PUT** /computeFitnessClear Compute and output a fitness value v

info ^

- GET** /nodes Get Nodes v
- GET** /status Get Status v

secret-shared ^

- PUT** /sessionSecret Create a new session with secret-shared data v
- PUT** /computePopulationOrder Compute a population order v
- PUT** /computeClassification Classify a population v
- PUT** /computeClearing Compute a clearing v

Schemas v

Figure 5: Swagger UI interface description as provided by PE service.

3.2 Blockchain

3.2.1 Software Repositories

The blockchain application uses the Cosmos SDK and is written in Go. It is split into two different modules, one for the credit archive and one for the event protocol. It follows the general architecture recommended by Cosmos SDK with several different parts. Amongst others, there is one part which is responsible for creating transactions, another one for processing them. Furthermore, the blockchain application itself is bundled with a web server for querying the data of the two modules.

To follow the best practices for Tendermint applications with the Cosmos SDK, Ignite CLI² was used. Ignite CLI provides project templates and facilitates code generation based on data structures defined with protocol buffers³. The templates and generated code also include a test suite for the different modules. Furthermore, Ignite CLI can be used during the development for loading code changes while the application is running without the need of restarting it manually. All the manual code extensions follow the same practices and style as the template and generated code, resulting in a uniform code base that is well structured and easy to extend and to maintain.

Additionally, there is another web server written in Scala 2.13, that utilises the Play Framework and provides a convenient write access based on online transaction signing. Online transaction signing provides only medium security but makes the integration in the proof-of-concept very easy. As the architecture is very modular, changing the way of writing and signing transactions for production would only require moderate effort.

The four nodes of the proof-of-concept are pre-configured and provided as Docker images based on Debian 11.

The software has been made available as open source on the AIT software site dedicated to SlotMachine⁴ and can be downloaded from <https://git-service.ait.ac.at/sct-slotmachine-public/blockchain-public>.

Additionally, pre-built containers are available from <https://git-service.ait.ac.at/sct-slotmachine-public/privacyengine-container>.

² <https://github.com/ignite/cli>

³ <https://developers.google.com/protocol-buffers>

⁴ <https://git-service.ait.ac.at/sct-slotmachine-public>

3.2.2 Installation and Deployment

For a quick deployment via Docker a docker-compose.yml is provided. For running the containers, it is sufficient to execute “docker compose up”. Each of the four nodes provides access via three ports.

26657-26660: allows low-level Tendermint queries, like requesting the block height or the transactions contained in a given block

1317-1320: allows higher-level Tendermint/Cosmos SDK queries for querying the data of the two modules

9000-9003: provides write access for credits and for events

3.2.3 Usage

The main ways for other software components to interact with the trustworthy storage is via the API for reading and for writing. On Tendermint, the procedures for reading and writing are very different. While reading is a relatively simple operation, writing requires the creation of a transaction including the cryptographic aspects, the distribution to the other nodes and, finally, the common agreement on a block that contains the transaction. For this technical reason, the API is split up and available via two different web servers running on two different ports. A Swagger description is also available. The read and write requests for the credit archive are already described in D2.2. The requests for the event protocol are described in chapter 4 of this document.

4 Specification Update

4.1 Introduction

In this chapter we present some extensions and minor modification to the specification presented in D3.2 [1]. However, it does not include any significant changes and does not ask for a new version of D3.2, it can be seen as a small amendment to it.

4.2 Privacy Engine

During the final integration phase we decided to change the clearing in such way as to return aggregated values for each AU. For this it was necessary to change the setup method to also take a “mapping” parameter that specifies which flight belongs to which AU.

In order to compare different ways of calculating and working with aggregated fitness values, we implemented two additional methods that work both differently from an MPC perspective, but also have different characteristics from the point of view of the Heuristic Optimizer component.

The first new method (`/computeClassification`) returns:

- an unordered list of indices of all configurations whose fitness values were above a threshold of 75% of maximum value
- The index of the configuration with the highest fitness value
- A truth value indicating whether this highest fitness value is greater than or equal to the previous highest fitness value

The second new method (`/computeFitnessClear`) just returns the computed fitness values in plain text. This API endpoint was introduced for testing and integration purposes and must be removed in real deployments.

4.3 Blockchain

In addition to the module for the credit archive there is a module for event protocol. There are several different events that allow a basic view on what is happening at the moment and what happened in the past in the system. These events share some common fields that are defined by the API and some further fields that are not specified directly by the API. The latter ones can be changed more easily as this would not require any modifications of the code of the blockchain application.

4.3.1 Event Description

JSON is used as the general data format. There are several fields that can describe an event, where only two of them have to be given for every event and the rest of the fields depend on the type of the event. All fields of an event are of type String. The field `payload` is a string encoded JSON object (i.e., with escaped quotation marks).

Obligatory Fields

- eventType
- timestamp (example: "2022-06-10T12:00:00Z")

Optional Fields

- airspaceUserId
- regulationId
- regulationType
- optimizationSessionId
- airportId
- payload

The table below shows which fields are currently used for each event type. It is not enforced by the blockchain application that these fields are present, because it should not be necessary to update the blockchain application if this specification changes.

eventType	timestamp	airspaceUserId	regulationId	regulationType	optimizationSessionId	airportId	payload
REGISTRATION_ADDED	X	X		X		X	callbackEndpoint
REGISTRATION_REMOVED	X	X		X		X	callbackEndpoint
NMF_CONNECTION_ESTABLISHED	X						nmfHost
REGULATION_ADDED	X		X	X		X	
OPTIMIZATION_INITIALISED	X		X	X	X	X	optimizationFramework, fitnessMethod
OPTIMIZATION_STARTED	X		X	X	X	X	
OPTIMIZATION_RESULTS_RECEIVED	X		X	X	X	X	solutionIds
VETO_RECEIVED	X	X			X		solutionId
SOLUTIONS_SUBMITTED	X				X		solutionIds
SOLUTION_ACCEPTED	X				X		solutionId

- REGISTRATION_ADDED, REGISTRATION_REMOVED: Specify events, which are added when registrations are either added or removed.
- NMF_CONNECTION_ESTABLISHED: This event is added after the first connection to NMF has been successful.
- REGULATION_ADDED: This event is added for each added regulation according to the flight list of the NMF.

- `OPTIMIZATION_INITIALISED`, `OPTIMIZATION_STARTED`, `OPTIMIZATION_RESULTS_RECEIVED`: These events are added when the optimization is initialised, started or finished. The event `OPTIMIZATION_RESULTS_RECEIVED` contains a list of possible solutions as its payload.
- `VETO_RECEIVED`: This event is added for each received veto.
- `SOLUTIONS_SUBMITTED`: This event is added after the solutions are submitted to the NMF.
- `SOLUTION_ACCEPTED`: This event is added after the accepted solutions has been communicated by the NMF.

4.3.2 API Endpoints

POST /add_event

The used ports for writing on the four nodes are 9000-9003, respectively. An event in JSON notation as described above must be added to the body of the request.

GET /event

The used ports for writing on the four nodes are 1317-1320, respectively. Analogous to the credits, there is a set of parameters concerning the pagination and the number of results per page. The parameter for reversing the order to list the newest events first is particularly important since these events are most of the time the most relevant.

- `pagination.reverse=true`

It is also possible to filter for some of the fields that are available for events. This allows some basic queries on events, but it is just possible to search for exact matches. It is not possible to retrieve some results based on partial hits, wildcards, or other advanced querying methods. If multiple fields are present in a query, the results are only those events where all the given values match the entries of the event.

- `eventType`
- `airspaceUserId`
- `regulationId`
- `regulationType`
- `optimizationSessionId`
- `airportId`

5 Security and Scalability

5.1 Privacy Engine

5.1.1 Security Analysis

The SlotMachine approach for privacy-preserving optimization of the assignment problem is based on the idea of splitting the computation into two interactive parts, one which is done obliviously on sensitive data and another part which is done in the clear. The performance advantage achieved compared to fully oblivious implementation of the Hungarian method is substantial and results from the reduction of computations done in the encrypted domain, which are time-consuming and typically slower by orders of magnitudes. By carefully tailoring the optimization tasks into plaintext operations done in the clear and oblivious operations done on the ciphertext, substantial performance improvements can be achieved.

Performing certain operations in clear comes at the price of information leakage; only specific algorithms allow for this kind of partitioning. Heuristic optimization as used in our application turned out to be well suited. The challenge was to tailor the optimization algorithms in a way that they work with only minimum information to prevent attackers from compromising the private inputs (weights). Attackers having access to PE communication must not be able to recover individual weights and even for the Heuristic Optimizer the privacy property should hold.

In fact, ideally the PE only reveals relations between solutions, i.e., an ordering of solutions in the population, but no absolute quality parameter. Even with such limited information the Heuristic Optimizer is able to conduct privacy-preserving optimization with outstanding performance.

The privacy of inputs is governed by two facts. First, provable secure MPC protocols in the PE allow computations in a fully oblivious way. Second, the PE interface guarantees that the Heuristic Optimizer is only revealing information about the ordering of a population of correct swaps. The approach is closely related to the concept of order preserving and *order revealing encryption*, which also reveals the order between ciphertexts. In our case we do not even leak ciphertexts, which is even better. Thus, the PE serves as an oracle only revealing the ordering of swaps and, therefore, the privacy property also holds in our setting. In addition, in the case of classification, the definitive order of a population is also hidden, which further increases the difficulty for an attacker.

Interestingly, the problem of recovering weights from PE queries turns out to be impossible even if the PE is also revealing the fitness values in clear. This is due to the fact that if an attacker knows the fitness for all possible $n!$ slot permutations for a problem instance with n flights and n slots, i.e., n^2 weights, it is not able to solve the corresponding system of equations for the weights. This is due to the special nature of the assignment problem which requires a one-to-one mapping of slot to flights and only allows for column permutations in the weight matrix. Therefore, it is important that the PE only answers correct swaps, where each flight is assigned to exactly one slot.

n	n^2	$n!$	$rank(\Pi)$
3	9	6	5
5	25	120	17
7	49	5040	37
9	81	362880	65

Table 2: Rank of equation system given by the set of all $n!$ possible swap permutations for given problem size n with n^2 unknown weights.

In Table 2 we show the calculated rank of the system of equations derived from querying the fitness of all possible permutations of a problem from the PE. The rank of the equation system is always smaller than the number of unknown weights (n^2). We calculated the rank for $n < 10$ and we expect this property to continue for larger n . This means that even if the PE would output individual fitness values an attacker would not be able to recover the weights directly. However, this assumption only holds if the weights are independent of each other, which is not always the case in our application. Input preferences from an individual participant typically will have a certain structure, e.g., the weights of slots for flights depend on margins and priority, which is additional information to be used in an attack.

Therefore, in our PE interface we offer both options to be used by the optimizer, which can dynamically choose the right algorithm depending on the security and performance requirements. In summary, it is possible to query the fitness values for allowed swaps as well as only the ordering or classification for the corresponding population of swaps.

5.1.2 Performance and Scalability

The performance of the final version of the PE component critically depends on the method chosen to compute the aggregated fitness values. In the discussion above we see that AU input privacy can be improved if only orders of swaps are revealed. However, this comes at a price as can be seen from a basic example in Table 3. Depending on the problem and population size as well as given reaction time different API calls can be used. As can be seen, the matrix size has a negligible impact, and total runtime is overwhelmingly determined by the size of the population.

	50x50 Pop 100	100x100 Pop 100	100x100 Pop 10
<code>/computeOrdering</code>	8.99 s	9.11 s	0.36 s
<code>/computeClassification</code>	2.05 s	2.11 s	0.26 s
<code>/computeFitnessClear</code>	0.08s	0.13 s	0.02 s

Table 3: Performance in seconds to compute output for given API calls to PE.

We would like to mention, that the measured values are true for the current implementation based on *MPYC*. The choice for *MPYC* was done early in the project when the security and problem instance was not fully defined and only `computeFitnessClear` was considered for implementation. However, with the advanced approach introduced in [9] we already showed that substantial improvements can be achieved also for ordering and especially classification when different protocols and a compiler based MPC system is used. We leave the switch to *MP-SPDZ* for the next iteration of

the PE as future work. Anyway, this is merely an engineering task, because we already showed feasibility and expected performance in our research paper.

5.2 Blockchain

5.2.1 Security

In blockchain-based systems security is already part of the design as they originated in the context of virtual currencies, where security is one of the biggest concerns. One part of the security concerns write access and in particular that no one should be able to write data in the name of another entity or modify data without having the appropriate permissions. This is realised with transactions and public-private key pairs that are used for signing. In the Tendermint ecosystem accounts, key pairs and signatures are not part of the most basic component, Tendermint Core, but are introduced as a part of the Cosmos SDK. So the security for writes relies on the concealment of the private keys and on honest nodes that validate all the signatures correctly.

For the read access, the story is a bit different, because the data is considered to be visible for all by default. This is a result of the high demand for transparency because it is hard to trust a distributed system without having the possibility to see what is going on. By running a private net, it is straightforward to allow read access only to involved parties. Protecting data of one user on the blockchain so that it cannot be read by others is a complicated topic as it is in conflict with other design aspects and requires some trade-offs. For now, we stuck to keeping the blockchain free of sensitive data by identifying information that can be disclosed in order to increase the level of transparency of the overall system.

In addition to its own read and write request, a user often relies also on the correct behaviour of others. This depends on the design of the blockchain application, the involved processes and the trustworthiness and incentives of the involved actors. In our design, AUs do not have to have a high level of trust in other AUs but have to have a rather high level of trust in the controller. The controller itself can be split up into several entities that are required to sign a transaction and thus vow for its correctness. While this is relatively simple from a technical perspective, it is doubtful if the respective roles can be sensibly assigned in our system. A more refined approach is to create some guarantees that the written data is correct by providing ZK-proofs and automatically check them in the blockchain application. This extension is not part of the proof-of-concept but might be implemented in the context of another project.

5.2.2 Scalability

As with other distributed systems, it is not feasible to scale a blockchain-based system arbitrarily in different dimensions. Basically, there are three dimensions: throughput, number of nodes and block finality. Pushing the limits of one dimension results most of the time in reducing the other limits. With our choice of Tendermint the time for agreeing on a block is by default 1 second and thus in comparison with other blockchain-based systems relatively low. When data is rapidly distributed, agreed on and finalised, users do not have to wait long for the transactions to be processed, and can be sure that already processed transactions cannot be invalidated at a later time. The fast block finality comes at the cost of throughput and the network size. In the proof-of-concept the throughput is about 10 transactions per second where there are still some possibilities to speed it up, but only to a limited extent. Tendermint uses a Byzantine-Fault Tolerant consensus protocol that requires each node to receive messages from more than $2/3$ of the nodes before being able to agree on one block. Thus, the



number of messages increases quadratic with the number of nodes. Also depending on other network properties, it is feasible to run Tendermint with several hundreds of nodes. None of these limits poses any real problem in our scenarios at the moment. But it is also important to note, that some possible extensions concerning the security can have a large impact on the performance, which can make it even more hard to find a suitable trade-off between the different properties of the system.



6 Conclusion and Outlook

We presented a quick overview of the final proof-of-concept implementation of the privacy-engine and blockchain component as delivered for demonstration of the SlotMachine platform. The implementations are also released as open-source software and all information to get started are presented in this document, thus, serving as introductory manual.

With the release of the final version of PE and BC we were able to achieve our goals and deliver core components to enable privacy preserving slot optimization as envisioned in SlotMachine. The implementation fulfils all requirements defined and reach an estimated maturity level of TRL3. TRL3 (experimental proof-of-concept) have been achieved given all the results achieved in the development of the privacy engine and blockchain integration as summarized below. We have shown practical performance and scalability for our first PE proof-of-concept implementation based on experimental data. Furthermore, we also demonstrated the use of blockchain and showed feasibility of extended features by standalone benchmarks. The current results present an ideal starting point for further research and development and to push the concept to higher TRL levels.

During our research and development of the components and system we successfully established the concept of fast and efficient privacy preserving optimization via heuristic algorithms, a very promising direction to achieve practically efficient systems. Based on recent scientific results we already developed the concept of the next generation of PE which will be able to further improve over the current version. Furthermore, we also identified open research challenges which we were not able to address anymore but can eventually be solved in future research efforts and would significantly improve on security, scalability, and transparency aspects. Thin the following we quickly discuss major gaps identified which should be closed in future work for a final product.

Privacy engine. With our approach we achieve practical performance, however, latest research results show that we can even go beyond with highly optimized implementations. Furthermore, the integration of the optimizer into the privacy-engine and the use of particular MPC-friendly heuristics would lead to further speedup. Moreover, currently we are limited to a small number of MPC nodes, to increase security we would also prefer MPC protocols which scale better in the number of nodes, additionally to the input size. In summary, more research should be done on the scalability of MPC system now the problem-solving strategy for slot ordering is clearer defined and can be particularly addressed in the development of the next evolution of the system.

Besides scalability, end-to-end verifiability turned out to be an important aspect for the platform, as it enables transparency through public online auditing. Although, the verifiability mechanisms were not integrated in the final PE, we already developed the concept and showed the feasibility for important steps in the SlotMachine process. More research is needed to make the full process verifiable, which is desirable from a user perspective and to run the application in a decentralized setting without any single point of trust. In summary, if we could make the full process publicly verifiable without sacrificing the privacy, this could tremendously increase the trustworthiness of the platform. Moreover, we would not need any trusted operator anymore and could run the optimization fully community based, which would be a completely new way to drive innovation in air traffic management.

Finally, we think that the use of additional (side) information would lead to better heuristics and optimization results. If we can learn from past situations and consider this in our optimization process, we can additionally improve on both ends, on platform level but also on the client side. However, because there will be always sensitive information involved, we propose the use of privacy preserving

machine learning to improve solution strategies and thus improve scalability and efficiency in the optimization step.

Blockchain. The credit balance can be a sensitive information if it is updated regularly, because it can reveal the preferences and the private value of an AU for particular slots. While our current solution is to shift the immediate updates off-chain, more elaborate techniques could be applied. There are cryptocurrencies that include different cryptographic techniques in order to obscure amounts associated to a transaction of an account and thus, to keep data secret. Tendermint does not have such mechanisms built-in, and it is not obvious how they can be integrated. Furthermore, such mechanisms would have a great impact on the blockchain applications and could potentially reduce the possible computations. But still, combining secrecy aspects with various computations on the blockchain could hold a lot of potential.

A different potential extension concerns the single actor that controls the credit balances. In the proof-of-concept every AU has to trust the controller in providing the correct values. The privacy engine already can compute ZK-proofs with which the results can be verified, but secure aggregation happens in the dedicated wallet manager. The currently implemented possibility to verify a result manually if needed is already a useful feature for an AU. However, it is worth investigating if the ZK-proofs can be directly included in the computation of the blockchain application so that the updates of credit balances always have to respect the results of the trading and could not be manipulated by the controller or wallet manager.

7 References

- [1] SlotMachine Consortium, „D3.2 Specification of the PrivacyEngine Component“, SlotMachine Report, 2022.
- [2] SlotMachine Consortium, „D2.2 System Design Document“, SlotMachine Report, 2021.
- [3] SlotMachine Consortium, „D2.1 Requirements Specification“, SlotMachine Report, 2021.
- [4] SlotMachine Consortium, „D4.2 Specification of Evolutionary Algorithms“, SlotMachine Report, 2021.
- [5] C. G. Schuetz, E. Gringinger, N. Pilon, und T. Lorünser, „A Privacy-Preserving Marketplace for Air Traffic Flow Management Slot Configuration“, in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 2021, S. 1–9. doi: 10.1109/DASC52595.2021.9594401.
- [6] T. Loruenser, F. Wohner, und S. Krenn, „A Verifiable Multiparty Computation Solver for the Assignment Problem and Applications to Air Traffic Management“. arXiv, 2022. doi: 10.48550/ARXIV.2205.03048.
- [7] S. Krenn und T. Lorünser, „Single-Use Delegatable Signatures Based on Smart Contracts“, in *ARES 2021: The 16th International Conference on Availability, Reliability and Security, Vienna, Austria, August 17-20, 2021*, 2021, S. 40:1—40:7. doi: 10.1145/3465481.3469192.
- [8] T. Lorünser., F. Wohner., und S. Krenn., „A Privacy-Preserving Auction Platform with Public Verifiability for Smart Manufacturing“, in *Proceedings of the 8th International Conference on Information Systems Security and Privacy - ICISSP*, 2022, S. 637–647. doi: 10.5220/0011006700003120.
- [9] C. G. Schuetz, S. Jaburek, K. Schuetz, F. Wohner, R. Karl, und E. Gringinger, „A Distributed Architecture for Privacy-Preserving Optimization Using Genetic Algorithms and Multi-Party Computation“, 2022, S. 1–18.
- [10] C. G. Schuetz, E. Gringinger, N. Pilon, und T. Lorünser, „A Privacy-Preserving Marketplace for Air Traffic Flow Management Slot Configuration“, in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 2021, S. 1–9. doi: 10.1109/DASC52595.2021.9594401.
- [11] SlotMachine Consortium, „D4.3 SlotSwapping System“, SlotMachine Report, 2022.

