D7.3

First version of the C3ISP platform and test bed

WP7 – C3ISP platform: Requirements / Architecture / Implementation and integration

C3ISP

Collaborative and Confidential Information Sharing and Analysis for Cyber Protection

Due date of deliverable: 30/09/2018
Actual submission date: 30/09/2018

30/09/2018
Version 1.0

Responsible partner: HPE
Editor: Mirko Manea
E-mail address: mirko.manea at hpe.com

Project co-funded by the European Commission within the Horizon 2020 Framework Programme		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

The C3ISP Project is supported by funding under the Horizon 2020 Framework Program of the European Commission DS 2015-1, GA #700294

Authors: M. Manea, P. Ciampoli, M. Russo (HPE), T. Nguyen, V. Herbert (CEA), I. Herwono (BT), R. de Lemos, D. Chadwick, J. Ziembicka, W. Fan (UNIKENT), F. Di Cerbo, J. Boehler (SAP), P. Mori, A. Saracino, G. Costantino, I. Matteucci (CNR), J. Dobos, C. Wong (3D REPO), R. Hamouda (GPS), S. Tranquillini, A. Arighi (CHINO)

Approved by: C. Fox (DIGICAT), P. Niamadio (GPS)

Revision History

Version	Date	Name	Partner	Sections Affected / Comments
0.1	22-Mar-2018	M. Manea	HPE	Initial ToC
0.2	23-Apr-2018	M. Manea	HPE	Revised ToC
0.3	27-Jul-2018	M. Manea	HPE	Improved ToC sections and merged first contributions
0.4	27-Aug-2018	M. Manea, P. Ciampoli, M. Russo, I. Herwono, S. Tranqullini, A. Arighi	HPE, BT, CHINO	Merged HPE, BT and CHINO contributions
0.5	04-Sep-2018	M. Manea, J. Dobos, R. de Lemos, D. Chadwick, J. Ziembicka, W. Fan, P. Mori, A. Saracino, G. Costantino, I. Matteucci, F. Di Cerbo	HPE, 3D REPO, UNIKENT, CNR, SAP	Merged 3D REPO, UNIKENT, CNR, SAP contributions
0.6	13-Sep-2018	M. Manea	HPE	Revised all contributions ready for internal review process
0.7	15-Sep-2018	C. Fox	DIGICAT	Revised document
0.8	18-Sep-2018	P. Niamadio	GPS	Revised document
0.9	20-Sep-2018	M. Manea, I. Herwono, T. Nguyen	HPE, BT, CEA	Addressed internal reviewers comments
1.0	30-Sep-2018	M. Manea	HPE	Final version

Executive Summary

The main objective of this deliverable is to describe the release of the C3ISP Demonstrator for the first version of the implemented C3ISP Framework, as well as of the test bed, due at Month 24.

First, we present the updated high-level architecture: some refinements have been done during the past twelve months, in particular the role of some components have been clarified (DPOS API and DSA Manager Gateway, to interact respectively with the DPOS and the DSA Manager subsystems) and some have been added (Buffer Manager used to create *Data Lakes* for running the analytics services).

The document contains a dedicated section to describe what has been released as software artefacts and describes precisely each software module, its pre-requisites, the installation procedures and operational guidelines.

We present also the methodology we used to assure that the first implementation of the C3ISP reference architecture is of high-quality: in fact, we laid out the testing activities guidelines for each component, as well as each partner responsibility.

The deliverable continues by illustrating the updates to each subsystem (Information Sharing Infrastructure – ISI, Information Analytics Infrastructure – IAI, DSA Manager, Common Security Services – CSS) and the status of the implementation and integration for the software.

We also describe the workflow of major C3ISP use cases, which have been refined with many details.

The document concludes with a description of the status of the C3ISP development and test bed environments, where we have deployed this first version of the framework.

Table of contents

1. Introduction

1.1. Overview

This document presents the first version of the reference architecture implementation for the C3ISP Framework, as it has been originally designed in D7.2. It also describes how the architecture evolved in the last year to address specific Pilots' requirements. The document describes also the released artefacts and the methodology used for testing the framework in a structured way. Finally, it provides updated data flow diagrams for the most important use cases provided by the C3ISP Framework (if changed with respect to original M12 architecture) and updates on the development and test best environments.

1.2. Deliverable Structure

The document is structured as follows:

- Section 2 is dedicated to illustrate the C3ISP architecture implemented for M24;
- Section 3 describes the software artefacts delivered for the first version of the C3ISP Framework;
- Section 4 shows the methodology used for testing in order to assure a high-quality software release;
- Section 5, 6, 7, 8 reports on the C3ISP subsystems updates respectively for ISI, IAI, DSA Manager and CSS, as well as their current implementation and integration status;
- Section 9 illustrates the fundamental C3ISP flows as they have matured during the last year;
- Section 10 describes updates we have done on the development and test bed environments used for implementation;
- Section 11 draws on conclusions and next steps;
- Section 12 onwards contain appendixes.

1.3. Definitions and Abbreviations

Term	Meaning
AES	Advanced Encryption Standard
C&C	Command and Control
C3ISP	Collaborative and Confidential Information Sharing and Analysis for Cyber Protection
CybOX	Cyber Observable eXpression
CI	Continuous Integration
CPE	Common Platform Enumeration
CSP	Cloud Service Provider
CSS	Common Security Services
CTI	Cyber Threat Information
CVE	Common Vulnerability and Exposure

CWE	Common Weakness Enumeration
DAST	Dynamic Application Security Testing
DDoS	Distributed Denial of Service
DMO	Data Manipulation Operations
DoW	Description of Work for Grant Agreement: 700294 — Collaborative and Confidential Information Sharing and Analysis for Cyber Protection (C3ISP)
DPOS	Data Protected Object Store
DPO	Data Protected Object
DSA	Data Sharing Agreement
FHE	Full Homomorphic Encryption
GDPR	General Data Protection Regulation (EU 2016/679), http://eur-lex.europa.eu/eli/reg/2016/679/oj
IAI	Information Analytics Infrastructure
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IP	Internet Protocol
ISI	Information Sharing Infrastructure
LTS	Long-Term Support
LOWMC	Low Multiplicative Complexity (a family of block ciphers)
MITRE	The MITRE Corporation, https://www.mitre.org/
NFR	Non Functional Requirement
NVD	National Vulnerability Database
OASIS	Organization for the Advancement of Structured Information Standards
OWASP	Open Web Application Security Project
OpenC2	Open Command and Control
MoSCoW	Must have, Should have, Could have, and Won't have but would like
Multiplicative depth	Multiplicative depth is the maximum number of multiplicative gates between an input and an output of the circuit
PKI	Public Key Infrastructure
PRINCE	64-bit block cipher with a 128-bit key optimized for low latency in hardware
Prosumer	An entity which is both a producer and a consumer of information, in particular of Cyber Threat Information
REST	Representational state transfer, a type of web services
RFI	Remote File Inclusion attack

SaaS	Software as a Service
SQLi	SQL injection attack
STIX	Structured Threat Information eXpression
TAXII	Trusted Automated eXchange of Indicator Information
TTP	Techniques, Tactics and Procedures
VCG	VisualCodeGrepper
VM	Virtual Machine
WAVSEP	Web Application Vulnerability Scanner Evaluation Project
XSS	Cross-Site Scripting attack

2. High-Level Architecture at M24

This section illustrates the current C3ISP Framework reference architecture as evolved at M24:

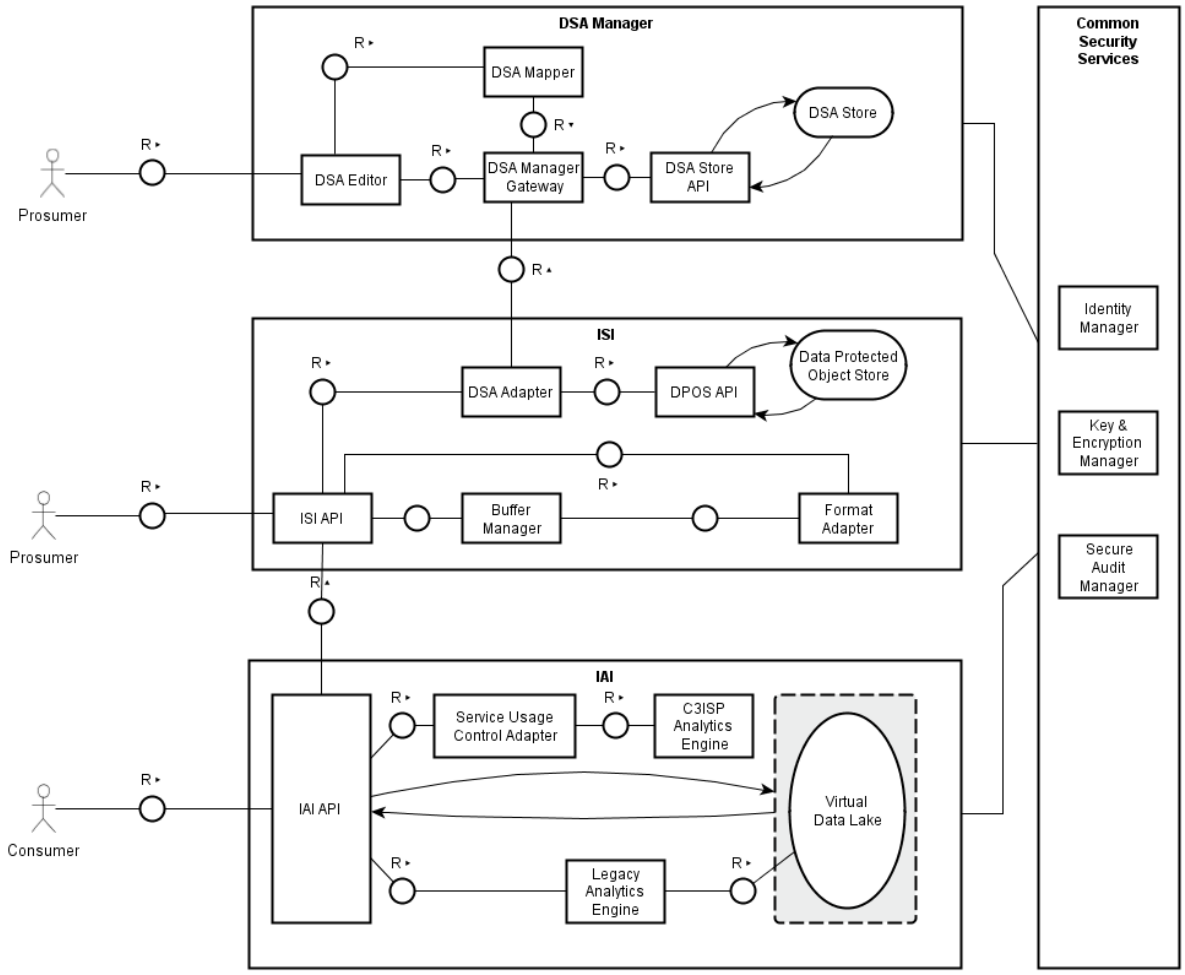


Figure 1: C3ISP high-level architecture – version Month 24

The C3ISP architecture is still composed by 4 major subsystems, whose role has not changed, namely:

- Information Sharing Infrastructure – ISI, devoted to providing sharing capabilities of CTI data regulated through sophisticated policies embedded in Data Sharing Agreements (DSAs);
- Information Analytics Infrastructure – IAI, responsible for regulating the execution of analytics services on the data shared through the ISI, hence governed by the DSAs;
- DSA Manager, for authoring the DSAs and handling their lifecycle, from initial writing in high-level language (CNL – Controlled Natural Language) to mapping in its enforceable representation (UPOL – Usage Policy Language[1]);
- Common Security Service – CSS, useful to provide services that operate across the other subsystems, like handling of identities and their authentication/authorization, encryption-related activities, and system auditing.

[1] UPOL is a low level directly enforceable XACML-based policy language, developed in the Coco Cloud FP7 project (grant no. 610853)

We have also clarified better some flows between the subsystems (see Section 9) which resulted in the following changes:

- In the ISI, we introduced the **Buffer Manager** component to be able to create data repositories where the analytics services can work to perform their function. These data repositories (or data lakes) are created as temporary data buffers that have a limited lifespan (e.g. the analytics execution), obey the DSA policies (e.g. DMOs like data anonymization are applied before putting the data on the "lake") and respect specific formats. The Buffer Manager interacts with the ISI API (see *prepareData* method exposed by ISI API in Section 5.6) and with the Format Adapter (to appropriately "format" the data lake for correct analytics consumption, see Section 5.2). This component is described in Section 5.3;
- In the ISI, we decoupled the interaction between DSA Adapter and the Data Protected Object Store[2] by adding a **DPOS API** component, which provides (and abstracts) the interface towards the DPOS. Previously, this role was in the component internal to the DSA Adapter called DPOS Connector, which has been removed. The DPOS API component is described in Section 5.4;
- In the DSA Manager, the DSA API has been split in two components, namely **DSA Manager Gateway** and **DSA Store API**. The DSA Store API is a CRUD-like interface to the DSA Store repository, while the DSA Manager Gateway exposes higher level API (like retrieve of the DSA state, which is a property of the DSA; or returning the UPOL representation of the DSA, which is a part of a mapped DSA) by interacting with the DSA Store API. These components are described respectively in Sections 7.1 and 7.3.

[2] In D7.2 the DPOS was called "Data Protected Object Storage"; in D7.3 we renamed it to "Data Protected Object Store" to have consistent names with the another store we have ("DSA Store")

3. Delivered C3ISP Software

This section describes the software artefacts released at **month 24**, their installation procedure with pre-requisites, and their operational procedures for usage/integration.

3.1. Software artefacts

The following depot containing all the C3ISP Framework artefacts are available for download[3]:

- https://devc3isp.iit.cnr.it/protected/C3ISP_Framework-M24.tar.gz

The table below lists the released software components contained in the depot. They are provided in packaged archives ready to be used for installation following the steps described in the next sub-sections.

Table 1 – Released software components for C3ISP Framework at M24

Subsystem	Component	Module	Filename
ISI	DSA Adapter	DSA Adapter Front End	ISI/dsa-adapter-frontend.war
ISI	DSA Adapter	Event Handler	ISI/event-handler.war
ISI	DSA Adapter	Continuous Authorization Engine	ISI/multi-resource-handler.war ISI/UsageControlFramework.war
ISI	DSA Adapter	Obligation Engine	ISI/trigger-engine.war ISI/action-engine.war
ISI	DSA Adapter	DMO Engine	ISI/dmo-engine.war
ISI	DSA Adapter	Bundle Manager	ISI/bundle-manager.war
ISI	Format Adapter	-	ISI/converter.js
ISI	Buffer Manager	-	ISI/buffer-manager.war
ISI	DPOS API	-	ISI/dpos-api.war
ISI	ISI API	-	ISI/isi-api.war
IAI	C3ISP Analytics Engine	-	IAI/iai-api.war[4] IAI/monitoring-dga.war
IAI	C3ISP Analytics Engine	FHE Analytics	IAI/fhe-conn-malicious-host.war
IAI	C3ISP Analytics Engine	Interactive 3D Visualization	https://github.com/3drepo/3drepo.io/archive/2.16.0.tar.gz URL: https://www.3drepo.io/[5]
IAI	Service Usage Control Adapter	-	*Not currently deployed*
IAI	Interface to Legacy Analytics Engines	-	*Uses IAI API*

[3] The access to this site is protected; credentials are delivered separately.

[4] This C3ISP release contains the C3ISP Analytics services bundled with the IAI API. It is planned to change in the next release.

[5] The access to this site is protected; request access for visualisations uploaded by other accounts

IAI	Virtual Data Lake	-	*This is implemented with an off-the-shelf product*
IAI	IAI API	-	IAI/iai-api.war
DSA Manager	DSA Editor	DSA Editor Front-end	DSA_Manager/DSAEditor.war
DSA Manager	DSA Editor	DSA Editor Tool	DSA_Manager/DSAAuthoringTool.war
DSA Manager	DSA Manager Gateway	-	DSA_Manager/DSAAPI.war
DSA Manager	DSA Mapper	-	DSA_Manager/dsa-mapper.war
DSA Manager	DSA Store	-	*This is implemented with an off-the-shelf product*
DSA Manager	DSA Store API	-	DSA_Manager/dsa-store-api.war
CSS	Identity Manager	-	*This is implemented with an off-the-shelf product*
CSS	Key and Encryption Manager	K&E Core	CSS/ke-core-manager-api.war
CSS	Key and Encryption Manager	DPO – Key & Encryption Manager	CSS/dpos-encryption.war CSS/dpos-key.war
CSS	Key and Encryption Manager	FHE – Key & Encryption Manager	CSS/fhe-encryption.war CSS/fhe-keys.war
CSS	Secure Audit Manager	-	*This is implemented with an off-the-shelf product*

3.2. Pre-requisites

The deployment of the C3ISP subsystems (ISI, IAI, DSA Manager, CSS) depends on the availability of hardware (or virtual hardware) and third-party software that must be installed. It is critical to install the pre-requisites to have a working C3ISP environment.

The versions reported in the next tables are those tested; generally, we do not foresee incompatibilities with newer versions.

Table 2 – Hardware requirements

Subsystem	Tested hardware	Operating System
ISI	2 vCPU, 6 GB RAM, 100 GB HDD	Linux Ubuntu 16.04 LTS
IAI[6]	40 CPU, 256 GB RAM, 6TB HDD	Linux Ubuntu 16.04 LTS
DSA Manager	2 vCPU, 4 GB RAM, 40 GB HDD	Linux Ubuntu 16.04 LTS
CSS	6 vCPU, 8 GB RAM, 20 GB HDD	Linux Ubuntu 16.04 LTS

[6] This powerful machine is physical and its hardware characteristics are required for FHE computation.

Table 3 – Software requirements

Subsystem	Third party software	Version	Description
ISI	Java JRE Apache Tomcat NodeJS Apache Hadoop HDFS Mongo DB	1.8.0_181 8.0.32 8.11.4 3.0.0 3.2.20	Java runtime engine Java servlet container to run C3ISP ISI modules Javascript engine to run Format Adapter DPOS Part of DPOS for DPOS metadata
IAI	Java JRE Apache Tomcat MySQL Apache Hadoop HDFS Cingulata[7]	1.8.0_181 8.0.32 5.7.23 3.0.0 N/A	Java runtime engine Java servlet container to run C3ISP IAI modules Internal DB for DGA; also act as VDL Virtual Data Lake (VDL) Cingulata compiler toolchain
DSA Manager	Java JRE Apache Tomcat MongoDB MySQL	1.8.0_181 8.0.32 3.2.20 5.7.23	Java runtime engine Java servlet container to run C3ISP ISI modules DSA Store Profile Store for DSA Editor
CSS	Java JRE Apache Tomcat HashiCorp Vault MySQL OpenLDAP Cingulata	1.8.0_181 8.0.32 0.10.0 5.7.23 2.4.42 N/A	Java runtime engine Java servlet container to run C3ISP ISI modules Key manager service Backend for Vault Identity Manager service Cingulata compiler toolchain

3.3. Installation procedures

The installation procedures are split for subsystem and described in the next sections.

3.3.1. ISI – Information Sharing Infrastructure

Most of the ISI released components runs on top of Apache Tomcat, which needs to be installed and running before proceeding with the ISI C3ISP installation.

The following ISI components are released as a Web Archive (.war) files; installation is straightforward and requires to upload the war files onto Tomcat. The Tomcat version shipped with Ubuntu Linux requires the war files to be copied to /var/lib/tomcat8/webapps.

Table 4 – ISI components (1)

Component	Path and file
DSA Adapter Front End	/var/lib/tomcat8/webapps/dsa-adapter-frontend.war
Event Handler	/var/lib/tomcat8/webapps/event-handler.war
Continuous Authorization Engine	/var/lib/tomcat8/webapps/multi-resource-handler.war /var/lib/tomcat8/webapps/UsageControlFramework.war
Obligation Engine	/var/lib/tomcat8/webapps/trigger-engine.war /var/lib/tomcat8/webapps/action-engine.war
DMO Engine	/var/lib/tomcat8/webapps/dmo-engine.war

[7] See D8.3 and https://github.com/CEA-LIST/Cingulata

Bundle Manager	/var/lib/tomcat8/webapps/bundle-manager.war
Buffer Manager	/var/lib/tomcat8/webapps/buffer-manager.war
Data Protected Object Storage	/var/lib/tomcat8/webapps/dpos-api.war
ISI API	/var/lib/tomcat8/webapps/isi-api.war

The following ISI component runs on NodeJS. It is required to deploy this on top of NodeJS; we use PM2 (Process Manager 2[8], a nodejs application runner):

Table 5 – ISI components (2)

Component	Path and file
Format Adapter	/home/nodejs/format-adapter/converter.js

3.3.2. IAI – Information Analytics Infrastructure

Most of the IAI released components run on top of Apache Tomcat, which needs to be installed and running before proceeding with the IAI C3ISP installation.

The following IAI components are released as a Web Archive (.war) files; installation is straightforward and requires to upload the war files onto Tomcat. The Tomcat version shipped with Ubuntu Linux requires the war files to be copied to /var/lib/tomcat8/webapps.

Table 6 – IAI components

Component	Path and file
IAI API	/var/lib/tomcat8/webapps/iai-api.war
FHE Analytics	/var/lib/tomcat8/webapps/fhe-conn-malicious-host.war
C3ISP Analytics Engine	/var/lib/tomcat8/webapps/monitoring-dga.war

3.3.3. DSA Manager

All DSA Manager released components run on top of Apache Tomcat, which needs to be installed and running before proceeding with the DSA Manager C3ISP installation.

The following DSA Manager components are released as a Web Archive (.war) files; installation is straightforward and requires to upload the war files onto Tomcat. The Tomcat version shipped with Ubuntu Linux requires the war files to be copied to /var/lib/tomcat8/webapps.

Table 7 – DSA Manager components

Component	Path and file
DSA Editor	/var/lib/tomcat8/webapps/DSAEditor.war /var/lib/tomcat8/webapps/DSAAuthoringTool.war
DSA Mapper	/var/lib/tomcat8/webapps/dsa-mapper.war
DSA Manager Gateway	/var/lib/tomcat8/webapps/DSAAPI.war
DSA Store API	/var/lib/tomcat8/webapps/dsa-store-api.war

[8] http://pm2.keymetrics.io/

3.3.4. CSS – Common Security Services

All CSS released components run on top of Apache Tomcat, which needs to be installed and running before proceeding with the installation.

The following components are released as a Web Archive (.war) files; installation is straightforward and requires to upload the war files onto Tomcat. The Tomcat version shipped with Ubuntu Linux requires the war files to be copied to /var/lib/tomcat8/webapps.

Table 8 – CSS components

Component	Path and file
Key and Encryption Manager	/var/lib/tomcat8/webapps/ke-core-manager-api.war /var/lib/tomcat8/webapps/dpos-key.war /var/lib/tomcat8/webapps/dpos-encryption.war /var/lib/tomcat8/webapps/fhe-key.war /var/lib/tomcat8/webapps/fhe-encryption.war

3.3.5. Deployment models

As described in D7.2, several deployment models are available. Pilots use the following C3ISP deployment models.

Table 9 – Deployment models in the Pilots

Pilot	Hybrid	Fully centralised
ISP Pilot	✓	
CERT Pilot	✓	
Enterprise Pilot		✓
SME Pilot	✓	

The fully centralised model has a single instance of ISI, IAI, DSA Manager and CSS. The installation procedure described above addresses the C3ISP fully centralised deployment model.

The hybrid mode has a local ISI and a centralised ISI, while keeping a single central IAI, as well as DSA Manager and CSS. For this reason, to install a hybrid C3ISP deployment we need to instantiate at least two ISIs, one acting as the central ISI part, the other(s) as local ISI(s). Considering a deployment with a single local ISI, we need to have a dedicate host (or virtual machine) where the ISI components will run, as reported in Table 4 – ISI components (1) and Table 5 – ISI components (2). With respect to the ISI pre-requisites (Table 3 – Software requirements), the DPOS will be configured to use a local filesystem, instead of the more resource-intensive and complex Apache HDFS, which is better suited for the central ISI. A "move DPO" operation transfers the created CTI bundle from the local ISI to the central ISI. Table 10 summarises the configuration changes for the local ISI deployment.

Table 10 – Configuration changes for local ISI deployment

Component	Description
DPOS	Configured to work on local filesystem instead of HDFS

ISI API	ISI API is configured to allow "create DPO" operation, only. Of course, that operation triggers the DSA policy evaluation at the local ISI, including enforcement of DMOs, if any. After the create DPO, ISI API should automatically invoke a "move DPO" operation. The move will transfer the CTI bundle from the local ISI to the central ISI. *At M24, the move operation is not automatically triggered and must be explicitly issued. Further, at this stage, bundle encryption keys are assumed to be the same between local and central ISI.*

3.4. Operational procedures

This section describes how the installed C3ISP Framework can be used/tested and how it can be integrated into/used by an application. To support these needs, C3ISP provides **external APIs** based on the RESTful web services paradigm (see D7.2, Section 2.1 where we described it), where there is a list of http endpoints that can be called programmatically. To easier testing, developers/testers can call the APIs through a simple web interface that describes their signatures and allows specifying the required parameters (these are based on Swagger Open API, as discussed in Section 4.2.5 and also in D7.2).

The external APIs are the only required to use the C3ISP functionalities and are used by the applications (i.e. Pilots). These APIs are the following:

Table 11 – C3ISP External APIs

Subsystem	Component	How
ISI	ISI API	RESTful webservices
IAI	IAI API	RESTful webservices
DSA Manager	DSA Editor	Web interface

The next sections briefly detail those interfaces. For more information, please refer to Sections 5.6 (ISI API), 6.5 (IAI API), 7.1 (DSA Manager) and the descriptions contained in D7.3.

3.4.1. ISI API

ISI API provides the functionalities of data sharing, including how to submit a CTI data to C3ISP and how to read it after.

APIs are documented via Swagger:

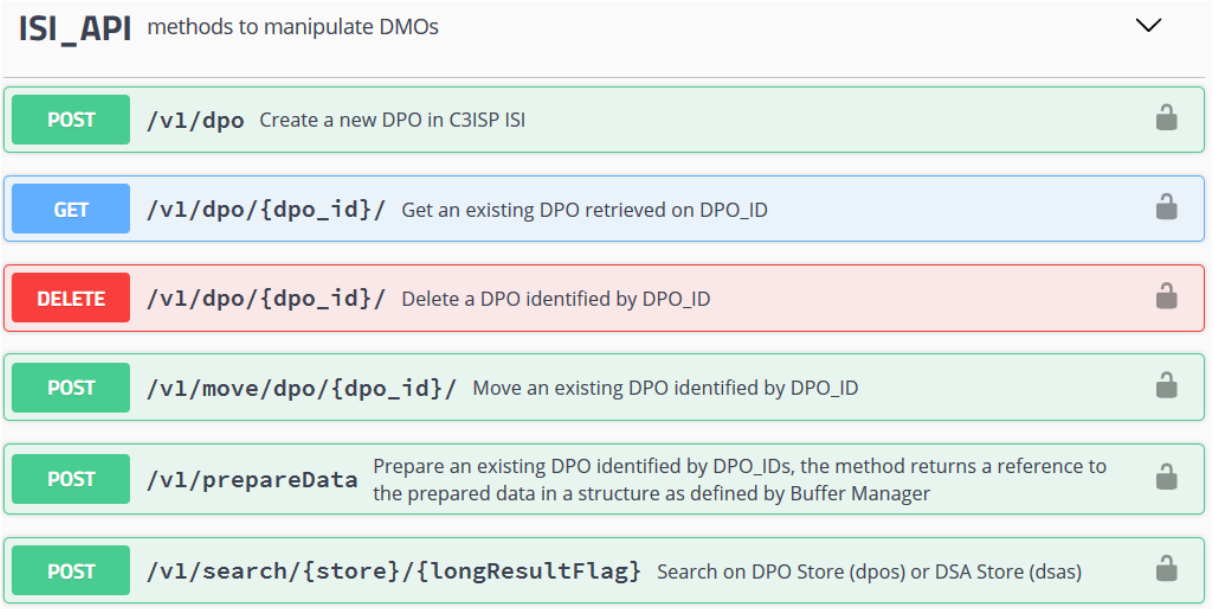


Figure 2: ISI API from Swagger UI

For example, to submit a CTI data to C3ISP, one needs to invoke a POST request on URL "/v1/dpo", passing as input an HTTP Form with two parameters:

- A file, with HTTP Form name = "fileToSubmit";
- A JSON string, with HTTP Form name = "inputMetadata".

The format of the JSON metadata can be depicted as follows:

```
{
 "Request" : {
  "Attribute" : [ {
   "AttributeId" : "ns:c3isp:dpo-metadata",
   "Value" : "{\"id\":\"4000123\",\"dsa_id\":\"DSA-56976731-3c16-46cc-a4e1-8384c6208eb0\",\"start_time\":\"2017-12-14T12:00:00.0Z\",\"end_time\":\"2017-12-14T18:01:01.0Z\",\"event_type\":\"Firewall Event\",\"organization\":\"3DRepo\"}",
   "DataType" : "string"
  } ]
 }
}
```

The return of such call is a JSON string, containing the DPO-Id associated to the file (if the request was successfully completed).

In order to read a CTI shared data, one needs to invoke a GET request on URL "/v1/dpo/<dpo-id>", where <dpo-id> must be replaced with the requested DPO-Id as received from the create

DPO call. Optionally, it is possible to submit as input a JSON string as HTTP header "X-c3isp-input_metadata" in the following format:

```
{
 "Request" : {
  "Attribute" : [ {
   "AttributeId" : <any desired metadata>,
   "Value" : <metadata value>,
   "DataType" : "string"
  } ]
 }
}
```

For further documentation, please refer to 5.6.

3.4.2. IAI API

IAI API provides the functionalities for running the analytics services on the data shared via ISI API.

APIs are documented via Swagger:

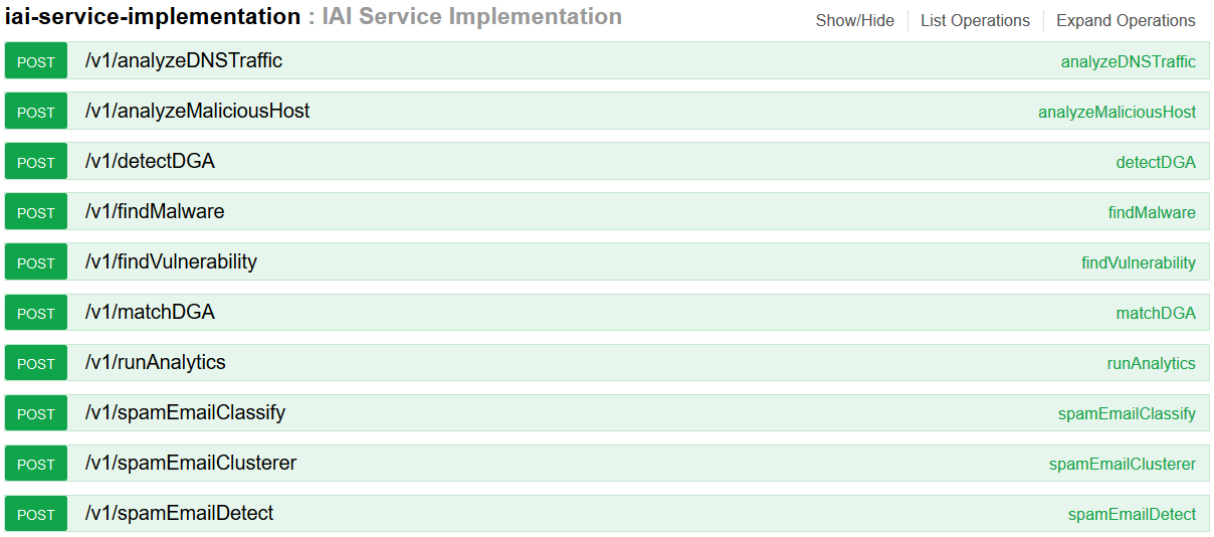


Figure 3: IAI API from Swagger UI

For example, to run an analytics service on a CTI shared data, one needs to invoke a POST request on URL "/v1/runAnalytics", passing as input an HTTP Form with the following parameter:

- A JSON string, with HTTP Form name = "param".

The format of the JSON structure can be depicted as follows:

```
{
 "metadata": {},
 "searchCriteria": {
```

```
  "combiningRule": "or",
  "criteria": [
    {
      "attribute": "event_type",
      "operator": "eq",
      "value": "DNS Query Request"
    }
  ]
 },
 "serviceName": "detectDGA",
 "serviceParams": {}
}
```


The return of such call is a JSON string, containing the DPO-Id associated to the result object (if the request was successfully completed).

The result is accessed via read API available in ISI API.

For further documentation, please refer to Section 6.5.

3.4.3. DSA Editor

DSA Editor is a web interface that allows defining the policies that regulate the CTI data sharing, including defining rules of access and usage control on shared data, on analytics services and results, and rules to handle data manipulation operations.

The following screenshots gallery presents the main DSA Editor screens. For further documentation, please refer to Section 7.1 and D8.2.

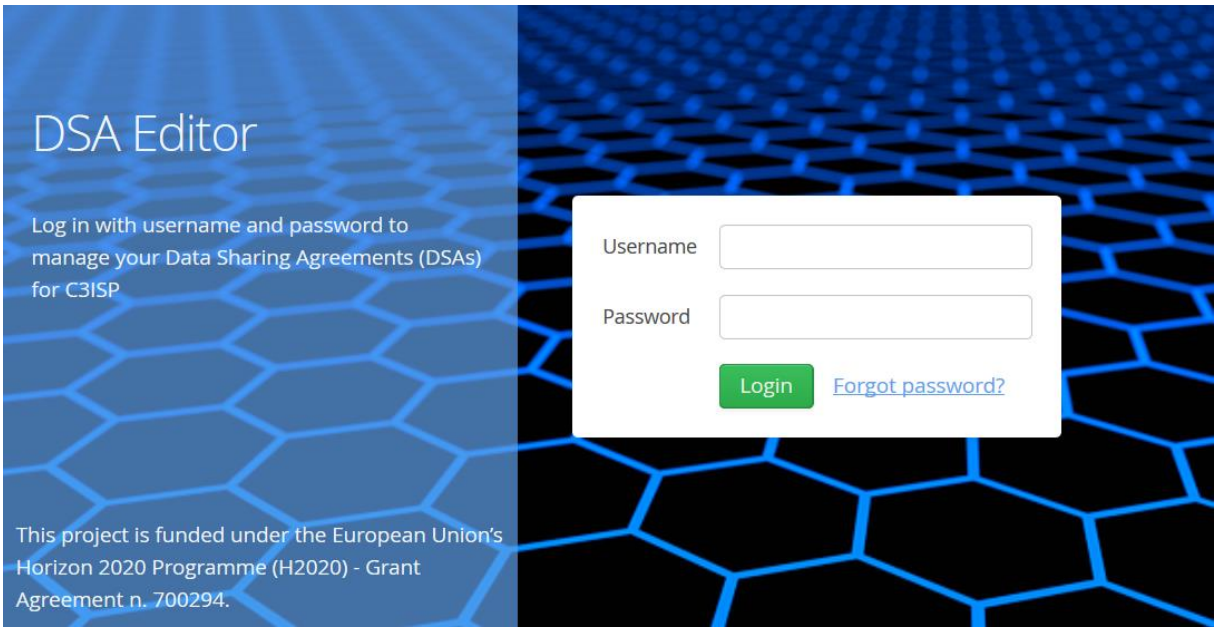


Figure 4: Login Screen

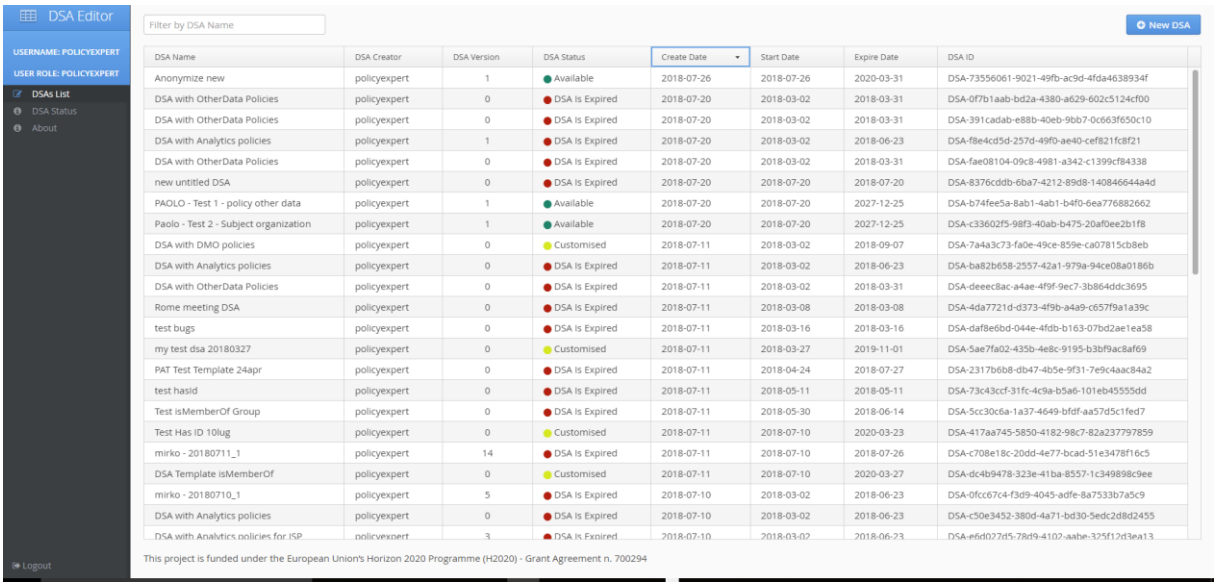


Figure 5: DSAs List

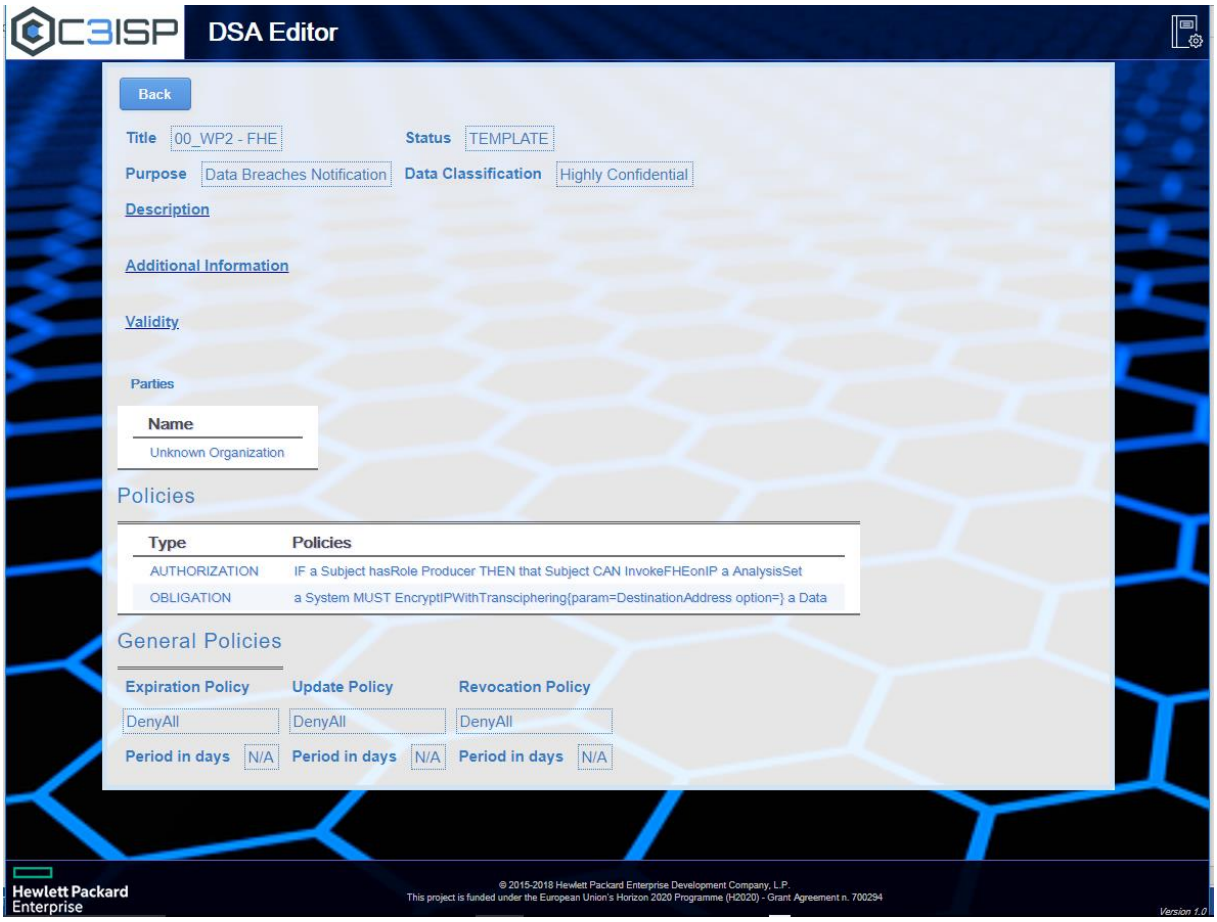


Figure 6: Show DSA

Figure 7: Edit DSA

4. First version of developed C3ISP platform

An important task of this work package (T7.3) is the system integration activities that aim to produce a working system from the distinct subsystems, components and modules developed for providing the C3ISP functionalities by means of the collaboration between the different consortium partners. The final goal is to have the implementation of the C3ISP reference architecture conceived in D7.2.

4.1. System Integration Methodology

The proposed integration process follows a step-by-step integration methodology to facilitate early identification and solution of integration problems. This process aims at mitigating the risk of failing the overall integration effort, by solving the major technical issues at an early stage.

The integration process comprises the following steps:

1. Identification of the communication interfaces;
2. Agreement about the responsibilities;
3. Definition of the communication interfaces;
4. Setup of the integration environment;
5. Point-to-point integration tests;
6. Multi-point integration tests;
7. Final system scenarios tests.

Collaboration between project partners, working remotely throughout Europe, has been fostered and improved by holding regular weekly conference calls, in addition to traditional e-mail communication using a dedicated mailing list. During the face-to-face meetings, we also always allocate dedicated slots for digging into the main issues.

4.2. Integration process

The following section illustrates how we setup the integration framework in order to successfully deliver the release of the C3ISP prototypes planned for M24.

4.2.1. Integration step 1 – Identification of the communication interfaces

C3ISP developers use unit tests for verifying the functionalities of the artefacts they are working on and further they perform integration tests at component level to check that every inner software module is properly working to implement the expected component behaviour.

The very first version of each component is tested using a black-box approach that includes feeding the component with input/output data from static files or stub programs and study the resulting behaviour at the borders. To integrate different components together we choose to have a network communication interface in order to exchange data at runtime. The communication between the different components and subsystems is reported in the architectural diagram in Figure 1, where the “links” show interacting components.

The following tables show per each subsystem and for each component the modules and the consortium partner that is developing it. When component has only one module, module name is not specified.

The next table is related to the **ISI** Subsystem:

Table 12 – ISI Subsystem: components, modules and partners

Component Symbol	Component Name	Module Symbol	Module Name	Owner
DSAAD	DSA Adapter	CAE	Continuous Authorization Engine	CNR
		OE	Obligation Engine	SAP
		DMO	DMO Engine	SAP
		EH	Event Handler	CNR/SAP
		DSAFE	DSA Adapter Front End	CNR/SAP
		BUNDM	Bundle Manager	CEA
FORM	Format Adapter	-	-	GPS
BUFM	Buffer Manager	-	-	CHINO
DPOS	DPOS	-	-	UNIKENT
DPOSAPI	DPOS API	-	-	UNIKENT
ISIAPI	ISI API	-	-	SAP

The next table is related to the **IAI** Subsystem:

Table 13 – IAI Subsystem: components, modules and partners

Component Symbol	Component Name	Module Symbol	Module Name	Owner
ANENG	C3ISP Analytics Engine	ANENG	C3ISP Analytics Engine	CNR
		FHEAN	FHE Analytics	CEA
		I3DVIS	Interactive 3D Visualisation	3D REPO
SUSC	Service Usage Control Adapter	-	-	CNR
LAS	Legacy Analytics Service	-	-	BT
VDL	Virtual Data Lake	-	-	BT
IAIAPI	IAI API	-	-	CNR

The next table is related to the **DSA Manager** Subsystem:

Table 14 – DSA Manager: components, modules and partners

Component Symbol	Component Name	Module Symbol	Module Name	Owner
DSAED	DSA Editor	DSAEFE	DSA Editor Front End	HPE
		DSAET	DSA Editor Tool	HPE
DSAMG	DSA Manager Gateway	-	-	HPE
DSAMAP	DSA Mapper	-	-	CNR
DSASTO	DSA Store	-	-	HPE
DSASTOAPI	DSA Store API	-	-	UNIKENT

The next table is related to the **CSS** Subsystem:

Table 15 – CSS: components, modules and partners

Component Symbol	Component Name	Module Symbol	Module Name	Owner
IDM	Identity Manager	-	-	CNR
K&EM	Key & Encryption Manager	KEC	K&E Core	CEA
		KEYM	Key Management	CEA
		DPOKEM	DPO - K&E Mng	CEA
		FHEKEM	FHE - K&E Mng	CEA
SECAUDM	Secure Audit Manager	-	-	HPE

Identity Manager and Secure Audit Manager could be integrated with external systems (off-the-shelf). For demonstrator, Identity Management is realized with an OpenLDAP (it could also be integrated with an Enterprise LDAP, like Microsoft Active Directory). Please note that some functionalities are provided by external tools, integrated via DMO Engine Plugin based architecture (see 5.1.5).

Table 16 – External systems integrated with C3ISP

Subsystem	Symbol	System	Owner
ISI	ANONT	Anonymization Toolbox	SAP
ISI	FHE	Full Homomorphic Encryption Toolbox	CEA
IAI	I3DVIZ	3D Repo Visualization Engine	3D REPO
IAI	SATURN	BT Saturn tool	BT
CSS	LDAP	Open LDAP	CNR
CSS	LOG	Secure Audit Tool	HPE

C3ISP has external end-to-end interfaces to allow input and output to/from the whole system for external actors (humans or systems). These interfaces are defined on specific components as shown in the following table:

Table 17 – C3ISP external end-to-end interfaces

Source	Destination	Components interacting
Prosumer	ISI	**Prosumer⇔ISI API** User interface (or an application on behalf of it) uses ISI API to share its data
Consumer	IAI	**Consumer⇔IAI API** User interface (or an application on behalf of it) uses IAI API to execute analytics jobs on data shared through the ISI
Prosumer	DSA Manager	**Prosumer⇔DSA Editor** User Interface uses services exposed by the DSA Editor to author DSAs

The communication **between the C3ISP subsystems** happens through well-specified points, as described in the following table (arrows indicate the direction where the communication starts):

Table 18 – C3ISP communication between subsystems (component[::module] notation used)

Source	Destination	Components interacting [::Module]
ISI	DSA Manager	DSA Adapter::Bundle Manager → DSA Editor::DSA Manager Gateway ISI uses DSA Manager Gateway to retrieve a DSA and its characteristics.
ISI	CSS	DSA Adapter::Bundle Manager → Key & Encryption Manager ISI uses the K&EM for encrypted operations on CTI data (bundle)
ISI	CSS	DSA Adapter::CAE/OE → Identity Manager CAE and OE connects to the IDM to retrieve attributes for policy evaluation
IAI	ISI	IAI API → ISI API IAI requests shared data for analytics processing and for data preparation in "data lakes"

In addition, each module/component of the three subsystems (ISI, IAI, DSA Manager) will trace its respective activities, for auditing purpose, to Secure Audit Manager (part of the CSS subsystem). Identity Manager is used by DSA Editor for user login and will be used for other internal authentication necessities, as well.

The communication interfaces **between the ISI components** can be summarised as follows:

Table 19 – ISI Subsystem: internal component[::module] communication

Components interacting	Description
ISI API → DSA Adapter::DSA Adapter Front End	Single communication entry point for using the DSA Adapter functionalities
DSA Adapter::Event Handler → Bundle Manager	Used for publish-subscribe of events related to the C3ISP bundle
DSA Adapter::Bundle Manager → Event Handler	Used for publish-subscribe of events related to the C3ISP bundle
DSA Adapter::Continuous Auth Engine → Event Handler	Used for publish-subscribe of events related to policy evaluation (usage control)
DSA Adapter::Event Handler → Continuous Auth Engine	Used for publish-subscribe of events related to policy enforcement (usage control)
DSA Adapter::Event Handler → DMO Engine	Used for publish-subscribe of events related to the execution of DMOs
DSA Adapter::Event Handler → Obligation Engine	Used for publish-subscribe of events related to obligation enforcement

DSA Adapter::DSA Adapter Front End → Event Handler	Kicks off the internal DSA Adapter workflow by publishing events to the Event Handler
DSA Adapter::Bundle Manager → DPOS API	Performs create/read/delete operation about C3ISP bundles
ISI API → Format Adapter	Requests CTI data format adaptation changes, if necessary
ISI API → Buffer Manager	Asks to prepare the data (and corresponding data lake) for analytics function execution
Buffer Manager → Format Adapter	Asks to update the CTI data format in such a way that the analytics service will be able to use it

The communication interfaces **between the IAI components** can be summarised as follows:

Table 20 – IAI Subsystem: internal components communication

Components interacting	Description
IAI API → Service Usage Control Adapter	IAI API checks whether the analytics it wants to run is authorised or not
Service Usage Control Adapter → C3ISP Analytics Engine	Mediates the access to the analytics services
Legacy Analytics Engine → Virtual Data Lake	Accesses the temporary virtual data lake where CTI data is stored for analysis
IAI API → Legacy Analytics Engine	Mediates access to the legacy services
IAI API → Virtual Data Lake	Allows accessing the VDL to setup it and properly format its content (thanks to interaction with Buffer Manager and Format Adapter)
Virtual Data Lake → IAI API	Allows accessing the VDL to setup it and properly format its content (thanks to interaction with Buffer Manager and Format Adapter)

The communication interfaces **between the DSA Manager components** can be summarized as follows:

Table 21 – DSA Manager Subsystem: internal components communication

Components interacting	Description
DSA Editor → DSA Mapper	Invokes the mapping services of the DSA from CNL language to UPOL language, to be ready to be enforced by the DSA Adapter component

DSA Editor → DSA Manager Gateway	Asks for services to persist DSA via the DSA Store API, by allowing advanced DSA management (e.g. DSA status check/change, etc.); keeps and manages the role-based access control settings for users accessing the DSA Editor
DSA Mapper → DSA Manager Gateway	Asks for services to persist DSA after mapping with the correct DSA state
DSA Manager Gateway → DSA Store API	Sends CRUD requests to stored DSA
DSA Store API → DSA Store	Simple CRUD interface towards the DSA Store

The communication interfaces **between the CSS components** can be summarized as follows:

Table 22 – CSS Subsystem: internal components communication

Components interacting	Description
Identity Manager → Secure Audit Manager	Traces Identity Manager activity
Key & Encryption Manager → Secure Audit Manager	Traces K&E Manager activity
Key & Encryption Manager → Identity Manager	Validates received requests based on the calling identity (requestor)
Secure Audit Manager → Identity Manager	Validates received logging requests based on the calling identity (requestor). Audit trails should come from authorised sources.

All the described communication flows are crucial to evaluate the correctness of the C3ISP Framework internal interactions and are used to perform the required integration testing activities.

4.2.2. Integration step 2 – Agreement about the responsibilities

A crucial step for a successful integration process is the agreement of the responsibilities for the definition, implementation and integration of the communication interfaces.

Roles and responsibilities of C3ISP partners for interface definition and integration, as enumerated in Step 1 (see tables in Section 4.2.1), have been agreed during conference call meetings and are reported in the following "RACI" matrixes (Table 24, Table 25, Table 26, Table 27, Table 28 and Table 29). The roles of each partner with respect to each communication interface are indicated in the next tables using abbreviations.

Table 23 – RACI acronyms definition

Acronym	Description
R	Responsible to perform the task (who does the work).
A	Accountable for the successful task completion.
C	Consulted before and during task execution.
I	Informed about task completion.

Table 24 – RACI matrix for communication with C3ISP (external interfaces)

Interaction	CNR	ISCOM-MISE	HPE	BT	SAP	CEA	DIGICAT	UNIKENT	GPS	CHINO	3D REPO
Prosumer⇔ISI API	**A**	I	C	C	R	I	I	C	C	C	C
Consumer⇔IAI API	**A, R**	I	C	C	C	I	I	C	C	C	C
Prosumer⇔DSA Editor	C	I	**A, R**	I	C	C	I	C	I	I	I

Table 25 – RACI matrix for communication between C3ISP subsystems

Source	Destination	Components interacting [::Module]	CNR	ISCOM-MISE	HPE	BT	SAP	CEA	DIGICAT	UNIKENT	GPS	CHINO	3D REPO
ISI	DSA Manager	DSA Adapter::Bundle Manager → DSA Editor::DSA Manager Gateway	C	I	C	I	C	**R**	I	**A**	I	I	I
ISI	CSS	DSA Adapter::Bundle Manager → Key & Encryption Manager	**R**	I	C	I	C	**A**	I	I	I	I	I
ISI	CSS	DSA Adapter::CAE → Identity Manager	**R, A**	I	C	I	C	I	I	I	I	I	I
ISI	CSS	DSA Adapter::OE → Identity Manager	A	I	C	I	**R**	I	I	I	I	I	I
IAI	ISI	IAI API → ISI API	A	I	C	I	C	I	I	I	I	**R**	I

Table 26 – RACI matrix for communication internal to the ISI Subsystem

Components interacting	CNR	ISCOM-MISE	HPE	BT	SAP	CEA	DIGICAT	UNIKENT	GPS	CHINO	3D REPO
ISI API → DSA Adapter::DSA Adapter Front End	**A**	I	C	I	**R**	I	I	I	I	I	I
DSA Adapter::Event Handler → Bundle Manager	C	I	C	I	**R**	**A**	I	I	I	I	I
DSA Adapter::Bundle Manager → Event Handler	C	I	C	I	**A**	**R**	I	I	I	I	I

DSA Adapter::Continuous Auth Engine → Event Handler	**R**	I	C	I	**A**	C	I	I	I	I	I
DSA Adapter::Event Handler → Continuous Auth Engine	**A**	I	C	I	**R**	C	I	I	I	I	I
DSA Adapter::Event Handler → DMO Engine	C	I	C	I	**R, A**	C	I	I	I	I	I
DSA Adapter::Event Handler → Obligation Engine	C	I	C	I	**R, A**	C	I	I	I	I	I
DSA Adapter::DSA Adapter Front End → Event Handler	**R**	I	C	I	**A**	C	I	I	I	I	I
DSA Adapter::Bundle Manager → DPOS API	C	I	C	I	C	**R**	I	**A**	I	I	I
ISI API → Format Adapter	C	I	C	I	**R**	I	I	I	**A**	I	I
ISI API → Buffer Manager	C	I	C	I	**R**	I	I	I	I	**A**	I
Buffer Manager → Format Adapter	C	I	C	I	C	I	I	I	**A**	**R**	I

Table 27 – RACI matrix for communication internal to the IAI Subsystem

Components interacting	CNR	ISCOM-MISE	HPE	BT	SAP	CEA	DIGICAT	UNIKENT	GPS	CHINO	3D REPO
IAI API → Service Usage Control Adapter	**R, A**	I	C	I	I	I	I	I	I	I	I
Service Usage Control Adapter → C3ISP Analytics Engine	**R, A**	I	C	I	I	C	I	I	I	I	C
Legacy Analytics Engine → Virtual Data Lake	C	I	C	**R, A**	I	I	I	I	I	I	I
IAI API → Legacy Analytics Engine	**R**	I	C	**A**	I	I	I	I	I	I	I
IAI API → Virtual Data Lake	**R**	I	C	**A**	I	I	I	I	I	I	I
Virtual Data Lake → IAI API	**R**	I	C	**A**	I	I	I	I	I	I	I

Table 28 – RACI matrix for communication internal to the DSA Manager Subsystem

Components interacting	CNR	ISCOM-MISE	HPE	BT	SAP	CEA	DIGICAT	UNIKENT	GPS	CHINO	3D REPO
DSA Editor → DSA Mapper	**A**	I	**R**	I	I	I	I	C	I	I	I
DSA Editor → DSA Manager Gateway	C	I	**R, A**	I	I	I	I	C	I	I	I
DSA Mapper → DSA Manager Gateway	**R**	I	**A**	I	I	I	I	C	I	I	I
DSA Manager Gateway → DSA Store API	C	I	**R**	I	I	I	I	**A**	I	I	I
DSA Store API → DSA Store	C	I	**A**	I	I	I	I	**R**	I	I	I

Table 29 – RACI matrix for communication internal to the CSS Subsystem

Components interacting	CNR	ISCOM-MISE	HPE	BT	SAP	CEA	DIGICAT	UNIKENT	GPS	CHINO	3D REPO
Identity Manager → Secure Audit Manager	**R**	I	**A**	I	I	I	I	C	I	I	I
Key & Encryption Manager → Secure Audit Manager	C	I	**A**	I	I	**R**	I	C	I	I	I
Key & Encryption Manager → Identity Manager	**A**	I	C	I	I	**R**	I	C	I	I	I
Secure Audit Manager → Identity Manager	**A**	I	**R**	I	I	I	I	C	I	I	I

4.2.3. Integration step 3 – Definition of the communication interfaces

A clear and detailed definition of the communication interface is required to have a smooth integration and to avoid rework.

Under WP7 supervision and proactive proposals, the responsible partners have refined or suggested the details for the implementation of the communication interfaces.

The fundamental interfaces have been described in D7.2 (First Version of C3ISP Architecture)

From a technical standpoint, whenever feasible, we chose to use the RESTful [1] web service paradigm for the communication interface. This is a lightweight and simple yet effective way to provide and consume web services.

The table below summarises the technical mechanism used for each communication:

Table 30 – List of interfaces and technical mechanisms

Interface	Technical mechanism	Documentation
Prosumer⇔ISI API	RESTful web services	5.6.2
Consumer⇔IAI API	RESTful web services	6.5.2
Prosumer⇔DSA Editor	Web browser (https)	7.1
DSA Adapter::Bundle Manager → DSA Editor::DSA Manager Gateway	RESTful web services	7.4.2
DSA Adapter::Bundle Manager → Key & Encryption Manager	RESTful web services	8.2.2.1
DSA Adapter::CAE → Identity Manager	LDAP	RFC 4511[9]
DSA Adapter::OE → Identity Manager	LDAP	RFC 4511
IAI API → ISI API	RESTful web services	5.6.2
ISI API → DSA Adapter::DSA Adapter Front End	RESTful web services	5.1.1.2
DSA Adapter::Event Handler → Bundle Manager	RESTful web services	5.1.6.2
DSA Adapter::Bundle Manager → Event Handler	RESTful web services	5.1.2.1

[9] Lightweight Directory Access Protocol (LDAP): The Protocol – https://tools.ietf.org/html/rfc4511

DSA Adapter::Continuous Auth Engine → Event Handler	RESTful web services	5.1.2.1
DSA Adapter::Event Handler → Continuous Auth Engine	RESTful web services	5.1.3.2
DSA Adapter::Event Handler → DMO Engine	RESTful web services	5.1.5.2
DSA Adapter::Event Handler → Obligation Engine	RESTful web services	5.1.4.2
DSA Adapter::DSA Adapter Front End → Event Handler	RESTful web services	5.1.2.1
DSA Adapter::Bundle Manager → DPOS API	RESTful web services	5.4.2
ISI API → Format Adapter	RESTful web services	5.2.2
ISI API → Buffer Manager	RESTful web services	5.3.2
Buffer Manager → Format Adapter	RESTful web services	5.2.2
IAI API → Service Usage Control Adapter	RESTful web services	6.2.1
Service Usage Control Adapter → C3ISP Analytics Engine	RESTful web services	6.1.3.1
Legacy Analytics Engine → Virtual Data Lake	LDAP/MySQL/HDFS	RFC 4511 / JDBC API [10] / HDFS API[11]
IAI API → Legacy Analytics Engine	RESTful web services Web browser redirects (https)	6.3
IAI API → Virtual Data Lake	MySQL/HDFS	JDBC API / HDFS API
Virtual Data Lake → IAI API	MySQL/HDFS	JDBC API / HDFS API
DSA Editor → DSA Mapper	RESTful web services	7.2.2
DSA Editor → DSA Manager Gateway	RESTful web services	7.4.2
DSA Mapper → DSA Manager Gateway	RESTful web services	7.4.2
DSA Manager Gateway → DSA Store API	RESTful web services	7.3.2
DSA Store API → DSA Store	RESTful web services	HDFS API
Identity Manager → Secure Audit Manager	RESTful web services	8.3.1
Key & Encryption Manager → Secure Audit Manager	RESTful web services	8.3.1
Key & Encryption Manager → Identity Manager	LDAP	RFC 4511
Secure Audit Manager → Identity Manager	LDAP	RFC 4511

4.2.4. Integration step 4 – Setup of the integration environment

The physical architecture of the C3ISP environment was described in D7.2 and now it has been updated in Section 10. The rationale behind that is the need to have a complete end-to-end integration test-bed as much as possible hosted in a single central environment, in order to facilitate the integration activities and the testing cycle. However, not all the components can

[10] Java JDBC AIP - https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

[11] Apache Hadoop Main API – https://hadoop.apache.org/docs/r3.0.0/api/index.html

run centrally. For example, depending on the C3ISP deployment model, ISI could run locally at Pilot premises (i.e. the hybrid model); in particular, for the sake of simplification, at this stage, we hosted also the on-premises part of the Pilots, including the local ISI deployment, for those that are using the C3ISP hybrid deployment model.

The integration environment aims at delivering the implementation of the C3ISP reference architecture conceived in WP7 and has the following logical roles:

- The C3ISP subsystems:
  - Information Sharing Infrastructure – ISI
  - Information Analytics Infrastructure – IAI
  - DSA Manager
  - Common Security Services – CSS (in particular K&E Manager is considered internal to the C3ISP Framework)
- External services: Anonymization Toolbox, FHE, CSS (Identity Manager and Secure Audit Manager are considered external off-the-shelf components)
- The four Pilots results: ISP, CERT, Enterprise, SME.

The environment is made mostly of virtual machines, so it is easy to reconfigure it in case it would be necessary for changing the allocation resources or cloning C3ISP artefacts to avoid bottlenecks in the testing activities. Only IAI runs on physical hardware, due to the high computational resources required by the (big data) analytics and by the homomorphic encryption.

The following table shows where each subsystem runs on which virtual machine:

Table 31 – C3ISP artefacts and corresponding machine

Roles	Artefact	Host
C3ISP Subsystems	ISI Subsystem	isic3isp.iit.cnr.it
	IAI Subsystem	iaic3isp.iit.cnr.it
	DSA Manager Subsystem	dsamgrc3isp.iit.cnr.it
	CSS Subsystem: Identity Manager (LDAP)	devc3isp.iit.cnr.it
	CSS Subsystem: Key Encryption	kec3isp.iit.cnr.it
	CSS Subsystem: Secure Audit Manager	<not yet implemented>
C3ISP Pilots	ISP Pilot	ispc3isp.iit.cnr.it
	CERT Pilot	90.147.82.10
	Enterprise Pilot	entc3isp.iit.cnr.it
	SME Pilot	smec3isp.iit.cnr.it

4.2.5. Integration step 5 – Point-to-point integration tests

The process of obtaining a working solution that delivers the planned functionalities requires several testing steps. The first step is to carry out point-to-point integration tests to verify each single communication interface. There are 35 interfaces as listed in Table 30 (4.2.3): these are also the number of point-to-point integration test suites needed (i.e. a set of tests for each interface).

In order to speed up integration and testing activities, and to proceed with testing even if components have not fully developed, we agreed to make use of some API mock services.

Swagger [1] is an open source solution that allows defining RESTful interfaces and automatically generates their documentation (in HTML format) together with the possibility of building requests for testing purposes (by using Swagger UI [3]). The generation of the HTML documentation and testing GUI is based on a YAML [5] file that has a specific Swagger syntax describing the interfaces exposed by the RESTful web service.

The figure below shows a sample of HTML documentation and testing GUI generated for the DPOS API component, with signature for input parameters and output values:

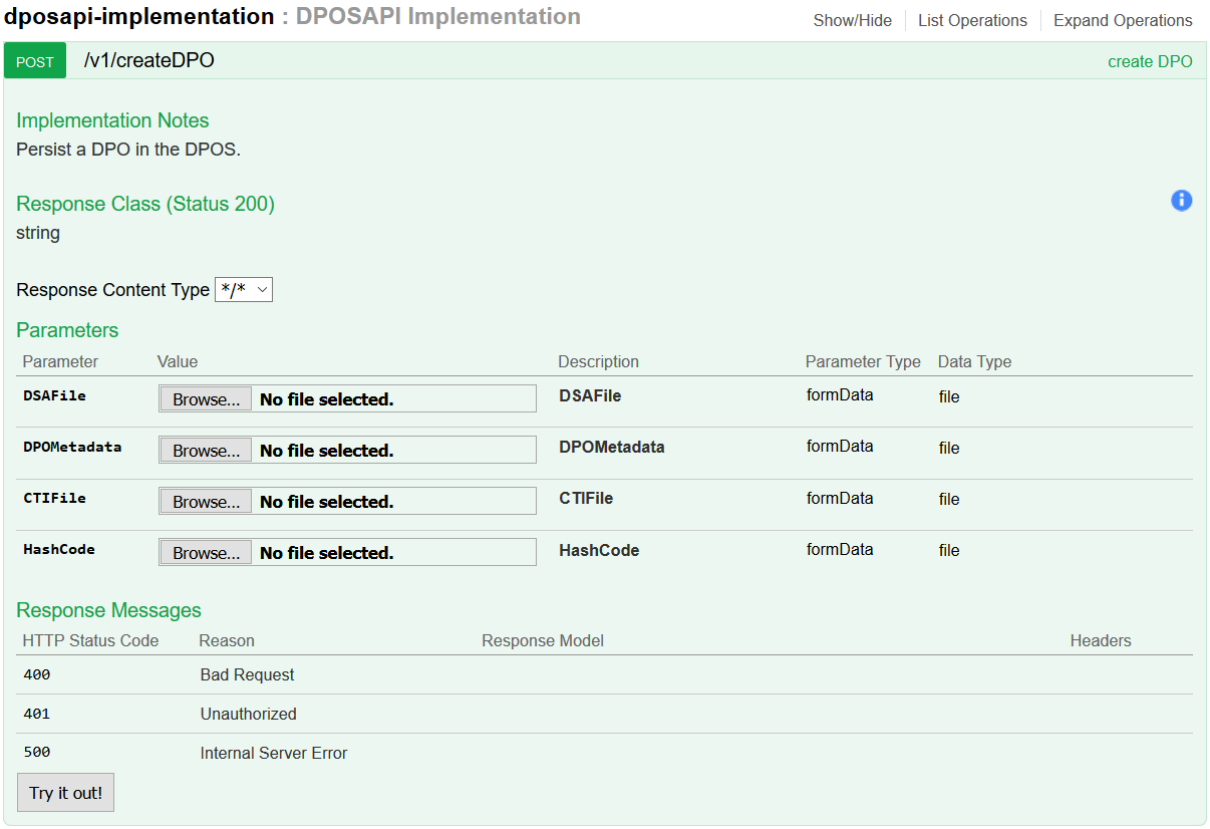


Figure 8: Auto generated documentation and testing GUI for a RESTful web service (Swagger UI). The "Try it out!" button allows generating a test web service request and checks the results.

The Swagger YAML file can be generated in several ways: automatically by using specific plugins, like SpringFox [6], that scan the application code in order to find interfaces definitions (i.e. described by Spring Boot [7] or JAX-RS [8] specific annotation), or through an Internet free portal service (called Swagger Editor [4]). Most of the C3ISP components' source code

uses Spring Boot to create RESTful web services, which are tagged with specific SpringFox java annotations to allow building the YAML specification (see Figure 9).

```
@ApiOperation(notes = "Persist a DPO in the DPOS.", value = "create DPO")
@ApiResponses(value = {
    //@ApiResponse(code = 201, message = "Created"),
    @ApiResponse(code = 400, message = "Bad Request"),
    @ApiResponse(code = 401, message = "Unauthorized"),
    @ApiResponse(code = 500, message = "Internal Server Error")
    })
@RequestMapping(method=RequestMethod.POST, value="/createDPO")
public ResponseEntity<String> createDPO(@RequestParam("DSAFile") MultipartFile dsaFile, @RequestParam("DPOMetadata") MultipartFile dpoMetadata,
            @RequestParam("CTIFile") MultipartFile ctiFile, @RequestParam("HashCode") MultipartFile hashCode) throws ParserConfigurationException, SAXException, IOException
        String resp;
HttpStatus httpstatus;
ResponseEntity<String> response;
        String ctiFilename = ctiFile.getOriginalFilename();
        LOGGER.info("cti File name : " + ctiFilename);

        byte[] dsaXMLBytes = dsaFile.getBytes();
        byte[] dpoMetadataBytes = dpoMetadata.getBytes();
        byte[] ctiBytes = ctiFile.getBytes();
        byte[] hashcodeBytes = hashCode.getBytes();
```

YAML generation

https://isic sp.iit.cnr.it/dpos-api/v2/api-docs

JSON Raw Data Headers

Save Copy Collapse All Expand All

```
swagger: "2.0"
info:
  description: "API for CRUD and Search operation on DPO"
  version: "1.0"
  title: "DPOS REST API"
  contact:
    name: "UniKent"
    url: "http://mysite/"
    email: "w.fan@kent.ac.uk"
host: "isic3isp.iit.cnr.it"
basePath: "/dpos-api"
tags:
  0:
    name: "dposapi-implementation"
    description: "DPOSAPI Implementation"
paths:
  /v1/createDPO:
    post:
      tags:
        0: "dposapi-implementation"
      summary: "create DPO"
      description: "Persist a DPO in the DPOS."
      operationId: "createDPOUsingPOST"
      consumes:
        0: "multipart/form-data"
      produces:
        0: "*/*"
      parameters:
        0:
          name: "DSAFile"
          in: "formData"
          description: "DSAFile"
          required: true
          type: "file"
```

Figure 9: Spring Boot Java code annotated with SpringFox (on the top) and its translation to YAML (on the bottom). Java annotation code starts with the @ character.

Having a Swagger UI that describes all the available API signatures and documentation, as well as that provides the capability to issue web services test requests, allows the setup of mock services. In fact, we first agreed on the list of API and their input/output parameters and built a service always returning fixed results: this allows proceeding in parallel to implement a workflow and not be blocked by waiting a component to be finished. Once the component mature and reaches stability[12], we replaces the fixed values with the real business logic.

[12] Every interface has a list of test cases prepared by the RACI responsible partner (**R**), who is also the partner responsible for executing the integration tests. Component stability is reached when at least one of the exposed

4.2.6. Integration step 6 – Multi-point integration tests

Testing the single point-to-point communication is not enough to have a working system. Components talks to each other in the context of the subsystem where they run (see Table 19 – ISI Subsystem: internal component[::module] communication, Table 20 – IAI Subsystem: internal components communication, Table 21 – DSA Manager Subsystem: internal components communication, Table 22 – CSS Subsystem: internal components communication), while subsystems interacts between them and with systems outside its perimeter via well-defined junction points (see Table 18 – C3ISP communication between subsystems). Multi-point integration tests require to develop an integration test plan that builds on already tested point-to-point interfaces, by adding incrementally a component after the other to define the full communication chain up to the complete testing of the end-to-end scenario.

In particular, multi-point integration tests start inside each subsystem (ISI, IAI, DSA Manager, CSS), then expand to the interaction between them and with the external systems (e.g. IDM).

For example, let us consider the DSA Manager subsystem: in order to test the interaction between the DSA Editor (DSAED) and the DSA Store (DSASTO), it is necessary that the point-to-point communications DSAED→DSAMG, DSAMG→DSASTOAPI and DSASTOAPI→DSASTO already work. Once we have tested those communications, we can proceed to expand the chain to include all of them to fully test the whole end-to-end scenario: DSAED→DSAMG→DSASTOAPI→DSASTO.

In this way, we create a test plan that enables us to incrementally test the interaction of the components by combining the point-to-point integration tests. The table below lists some of the incremental tests currently undergoing:

Table 32 – Multi-point integration sample testing scenarios

Components interacting	Description
1. Ext → ISI API → Format Adapter → DSA Adapter → DSA Manager → Key & Encryption Manager → DPOS API → DPOS 2. ISI API → DSA Adapter Front End → Event Handler → Bundle Manager → DSA Manager (DSA Manager Gateway)	1. Create DPO: a Producer asks to Create a DPO giving {metadata, CTI}. ISI normalizes CTI (Format Adapter), is authorised to create a DPO (DSA Adapter::Engines), creates the DPO-Id (DSA Adapter::DSA Adapter Front End), creates the bundle with DSA (DSA Adapter::Bundle Manager) by fetching DSA in UPOL format, encrypts CTI with a specific key, creates DPO that contains the CTI (with metadata) and the DSA (DSA Adapter::Bundle Manager), finally returns to the Ext (caller). 2. Fetch the right DSA/UPOL to be used: this is another relevant multi-point integration tests, which, even if it is included in 1, it is worth testing as standalone scenario. See workflow in Section 9.1.

API methods passes the internal integration test cases list: this would be better called “partial stability”, but our mock services approach allows us to have method-granularity control and replace a single mock method at a time.

Ext → ISI API → DSA Adapter → DPOS API → DPOS	Read DPO: a Producer asks to read a DPO. DSA Adapter checks policy evaluation and enforces (e.g. apply DMOs) the applicable DSA policy. See workflow in Section 9.2.
Ext → ISI API → DSA Adapter → DPOS API → DPOS	Delete DPO: ISI checks policy and removes DPO from DPOS. See workflow in Section 9.3.
1. Ext → ISI API → Buffer Manager 2. Buffer Manager → C3ISP Analytics Engine → ISI API → DSA Adapter 3. ISI API → Buffer Manager → ISI API → DSA Adapter → Virtual Data Lake(VDL)/Data Lake Buffer(DLB) → Format Adapter	Run Analytics: Prosumer runs Analytics Service giving as input {metadata, analytics service name, analytics service parameters, DPO search criteria}. Due to the complexity of the involved workflow, we split into many multi-point integration tests: 1. Creation of the data lake (VDL/DLB) considering Buffer Manager as a black box; 2. Execution of the analytics service and store of the result into C3ISP; 3. Creation of the data lake (VDL/DLB) with all the involved chain of components. See workflow in Section 9.4.

4.2.7. Integration step 7 – Final system scenarios tests

The last step is the testing of end-to-end realistic scenarios that evaluates input/output considering a black-box system. The entry points to the system are described in Table 17 – C3ISP external end-to-end interfaces. This activity leverages on the experience gained from the whole integration process testing so far. In particular, activities from Step 6 of the methodology can be combined to create system end-to-end test cases, by using real data and applications. In fact, Pilots can take advantage on the experience made to both perform testing activities and to spot quickly issues by isolating the affected components.

Please refer to Pilot Product deliverables (D2.3, D3.3, D4.3, D5.3) for further information on the Pilots end-to-end scenarios.

5. Subsystem Updates & Status: ISI – Information Sharing Infrastructure

The Information Sharing Infrastructure (ISI) allows Prosumers to exchange CTI data under the constraints specified in the DSA policies and acts as a storage of CTI data for access by analytics services in a controlled manner.

This section reports on the revised ISI that has been developed at M24 and deployed into the C3ISP Test Bed.

The ISI is made up of the following components, described in detail in the next sections:

- DSA Adapter: to enforce the rules written in the DSAs;
- Buffer Manager: to create a temporary data lake used by the analytics;
- Format Adapter: to accommodate for different necessities of data format and conversion;
- Data Protected Object Store (DPOS) and DPOS API: to securely persist the protected CTI data on a storage area for further sharing and processing;
- ISI API: to manage the external communication with the others C3ISP subsystems and users.

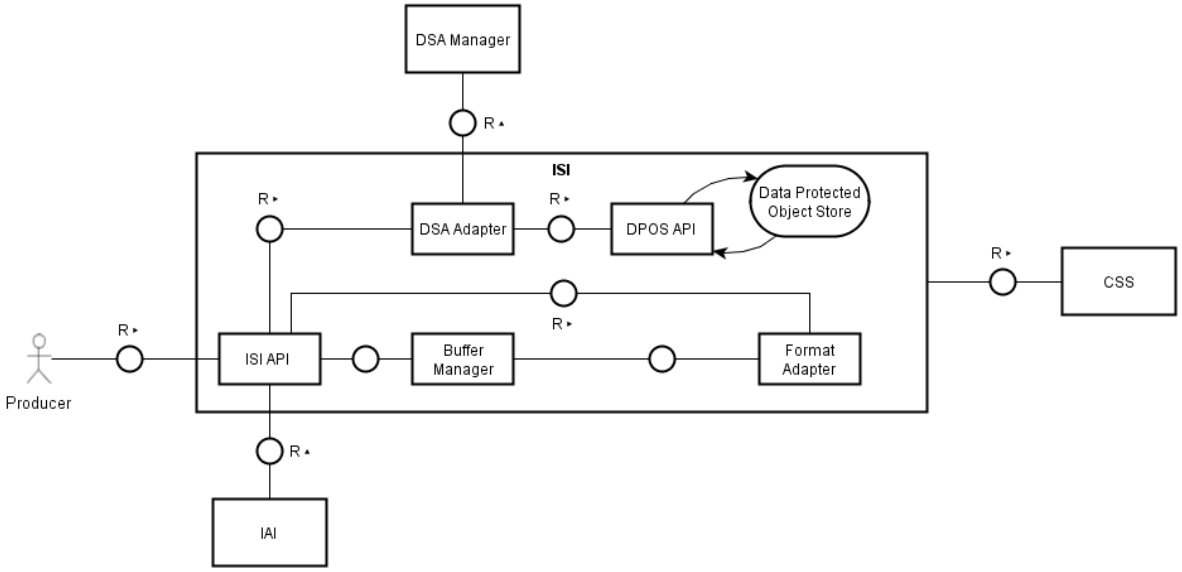


Figure 10: Information Sharing Infrastructure

The ISI interacts with external clients:

- The Producer (or an application on behalf of it), which would use the ISI API to share its data;
- The DSA Manager, for retrieving and evaluating the DSA used to protect the data;
- The Information Analytics Infrastructure (IAI) that will request shared data for analytics processing;
- The Common Security Services (CSS) for secure auditing of its activities, for identity management and for key and encryption services.

5.1. DSA Adapter

The **DSA Adapter** is the component of the C3ISP architecture which is in charge of evaluating the DSA paired with the CTI data and of enforcing it when the execution of some operation on the data is requested (e.g. read).

Figure 11 shows the modules of the DSA Adapter at month 12:

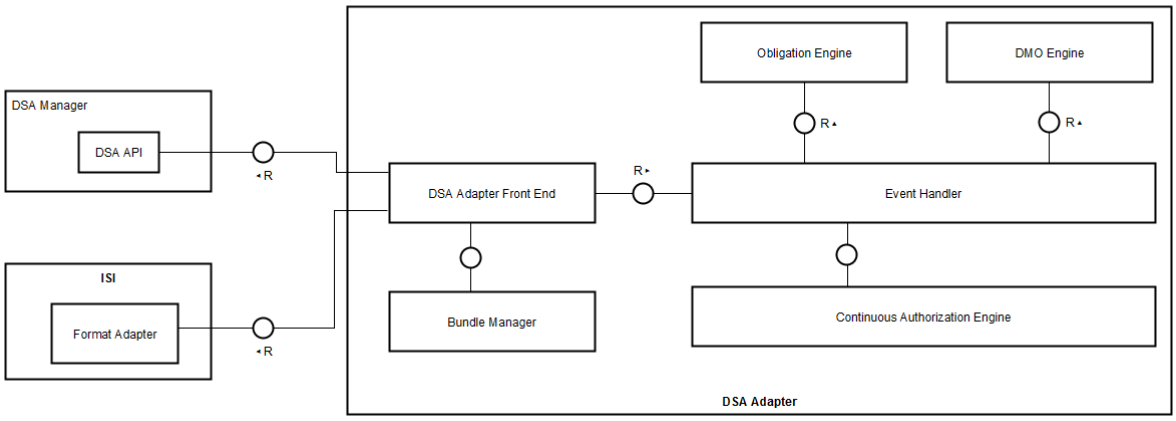


Figure 11: Components of the DSA Adapter at month 12

As shown in Figure 12, the diagram has been updated by moving the Bundle Manager to interact with the Event Handler, in such a way that now the Event Handler coordinates the main DSA Adapter modules in the same way, making the architecture cleaner and easier to be implemented:

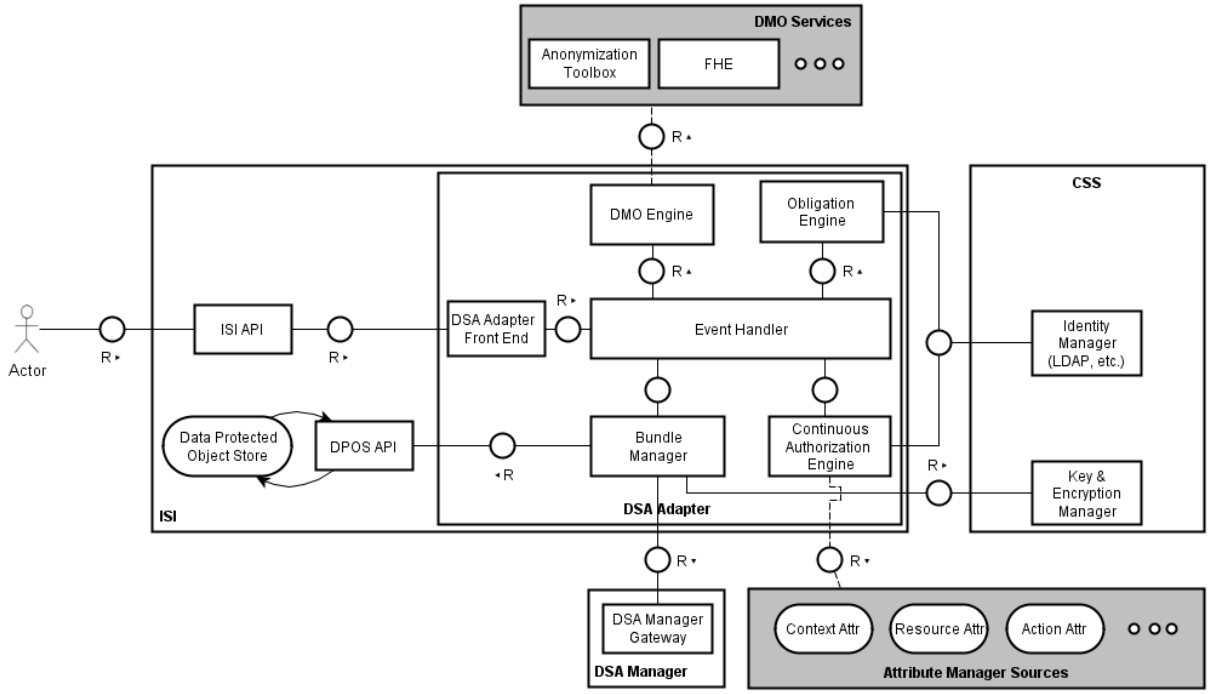


Figure 12: Components of the DSA Adapter at month 24

The new diagram shows also better the interaction with the ISI and CSS components. In fact, both the Continuous Authorization Engine and the Obligation Engine require attributes for policy evaluation from the Identity Manager, while the Bundle Manager selects the DSA from the DSA Manager and handles the encryption of the CTI data.

The following subsections describe the status of implementation for this First Version of the C3ISP platform (M24).

5.1.1. DSA Adapter Front End

The **DSA Adapter Front End** is the entry point of the DSA Adapter. It is in charge of exposing the component's public API consumed by the ISI API and of starting/tracking the business processes of the DSA Adapter needed to fulfil external requests.

Its architecture consists of:

- A module that contains the implementation of the exposed API;
- A module that communicates with the Event Handler and by means of the latter, with all other DSA Adapter components.

Any updates to the API will only occur provided they do not impact the architecture of the DSA Adapter Front End as shown in Figure 13:

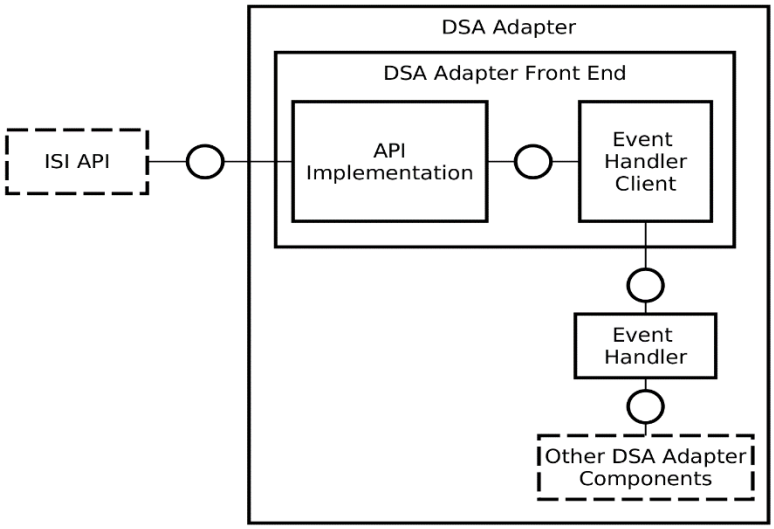


Figure 13: DSA Adapter Front End architecture

5.1.1.1. Implementation and Integration Status

At M24 the DSA Adapter Front End has been developed and is available as a RESTful web service. It is integrated with the ISI API and is able to interact with the Event Handler and indirectly with Bundle Manager and the Continuous Authorization and Obligation Engines.

The major goal for the final prototype is to fully support all functionalities exposed by the ISI API, avoiding any further architectural modification and with minimal (if any) updates to the exposed API. Improvements in the Move DPO functionality may be required and thus implemented.

5.1.1.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the DSA Adapter Front End:

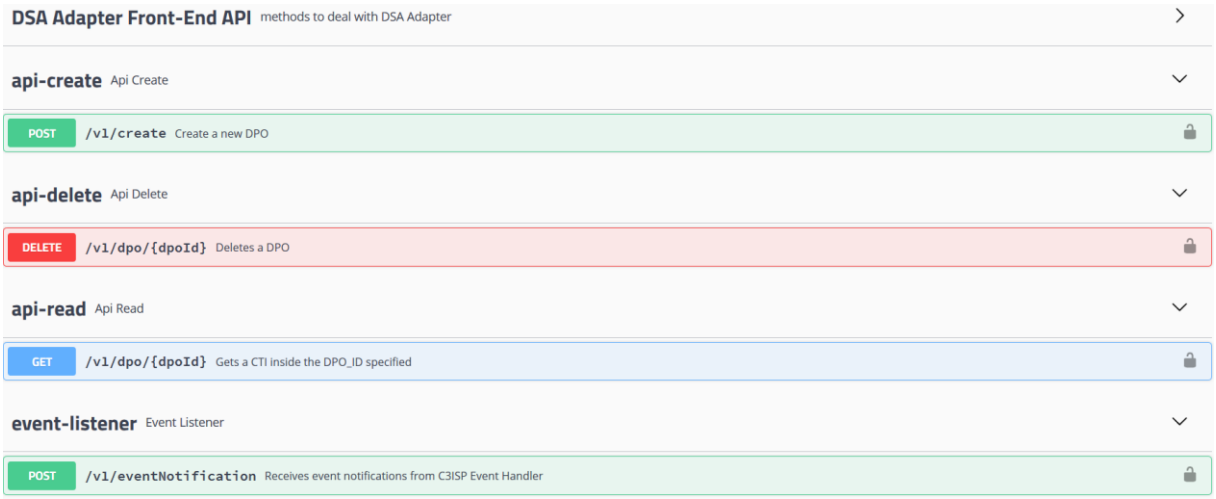


Figure 14: DSA Adapter Front End API from Swagger UI

5.1.2. Event Handler

The **Event Handler** is a component in charge of dispatching messages (events) to the DSA Adapter components. It implements a publish-subscribe message passing pattern.

At each start-up, the Event Handler requires other DSA Adapter components to register themselves to their messages of interests. It is designed to be the first element of DSA Adapter to be started, thus allowing the registration of other components. If a component faces an issue at its start-up, it will probably not be able to register to the Event Handler and this lack is easily detectable.

The Event Handler is also used to notify a message to one or more DSA Adapter components. When a new message has to be sent, its producer can do so by issuing a request to the Event Handler. The latter will proceed to notify each components (“push”) on a dedicated interface that all components need to expose and declare at the moment of the registration. Alternatively, the Event Handler offers to cache messages if a component prefers a “pull” notification model. In that case, it is that component’s responsibility to regularly query the Event Handler, that creates a dedicated message queue.

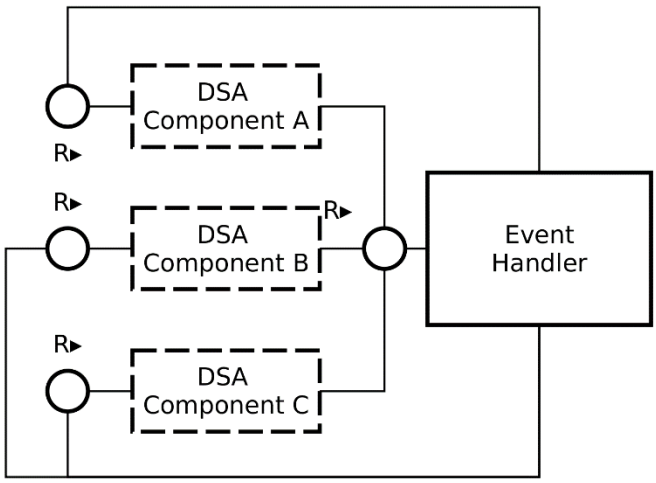


Figure 15: Event Handler architecture with interacting DSA Manager's components

The described architecture is to be considered in its final version, even if refinements to some of the exposed functionalities may still occur, if it will be deemed necessary.

5.1.2.1. Implementation and Integration Status

At M24 the Event Handler is fully available and its features are complete. No modifications are expected for the release of the final prototype. DMO Engine, Obligation Engine, Bundle Manager and Continuous Authorization Engine are integrated with the Event Handler.

5.1.2.1. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the Event Handler:

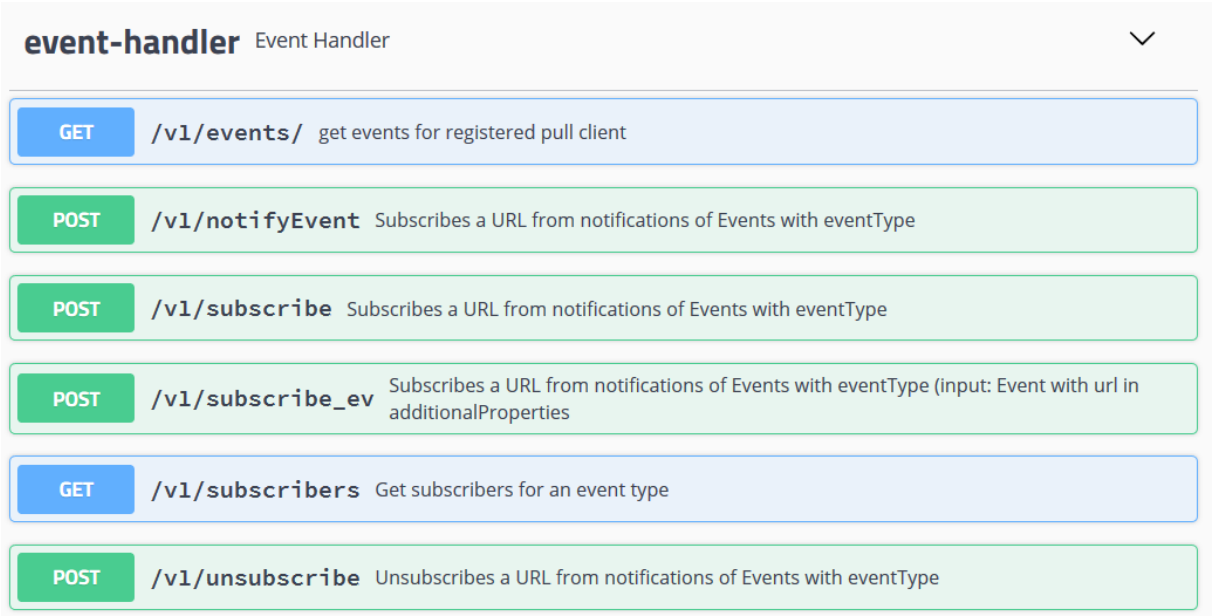


Figure 16: Event Handler API from Swagger UI

5.1.3. Continuous Authorization Engine

The **Continuous Authorization Engine** is the component of the ISI subsystem which is in charge of control the usage of CTIs, i.e., it is the component responsible of checking the usage

control policy paired with the CTI both at access request time (as in the traditional access control model) and continuously while the usage of the CTI is in progress, following the UCON model [22]. A detailed description of the internal architecture and of the features and functionality (shown in Figure 17) can be found in Deliverable D8.2.

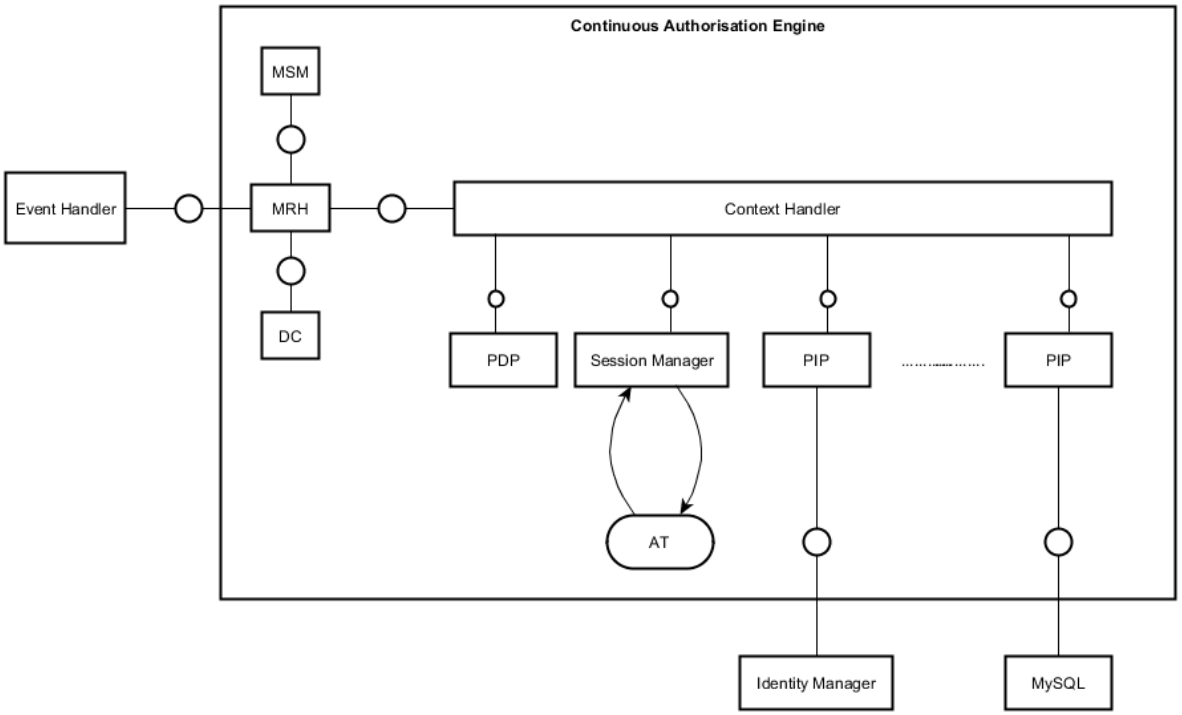


Figure 17: Internal Architecture of the Continuous Authorisation Engine

5.1.3.1. Implementation and Integration Status

In the following, we report a very brief description of the task of each of the Continuous Authorization Engine components, and a description of the main changes that have been performed at M24 to allow its integration within the ISI subsystem in order to perform usage control on the CTIs exploited in the C3ISP analytics.

In particular, the Continuous Authorization Engine consists of the following modules:

- **Multi-Resources Handler (MRH)**: This component has been introduced as result of the maturation process of the Continuous Authorization Engine component, in order to address the C3ISP pilots use cases. As a matter of fact, the original component was able to deal with access requests concerning one resource only, while in the C3ISP scenario the access requests concern sets of resources, i.e., the sets of CTIs on which the users want to execute the C3ISP analytics. Hence, with respect to the original version of the Continuous Authorization Engine (described in D8.1), the integration within the ISI subsystem required a modification of the original architecture, i.e., the introduction of this new component called Multi-Resources Handler, which intermediates the interactions between the Event Handler and the original Context Handler (CH). This component is in charge of exposing the new interface accepting multi resources access requests and of extending the workflow of the policy evaluation process to accommodate the evaluation of access requests involving multiple CTIs. In particular, this component interacts with the Event Handler accepting access requests involving multiple CTIs, perform a loop cycle to ask to the original Usage Control systems to evaluate the usage control policy of each of them (i.e., interacting with the original CH exploiting the original protocol), collects all the responses, and interact with the

Decision Combiner (DC) component to combine such responses in order to define the maximum set of CTIs for the analytics the be performed.

The MRH exposes the following APIs:

- **endAccessResponse**: invoked as part of the asynchronous communication between the Continuous Authorization Engine and the Event Handler. Reports if the session has been closed correctly with correct execution of the post obligations;
- **eventArrived**: API invoked to handle all events received from the event handler;
- **onGoingEvaluation**: invoked to check if the policy conditions still hold and eventually trigger a session revocation;
- **startAccessResponse**: confirms the execution of all operations related to the startAccess invocation;
- **tryAccessResponse**: invoked as part of the asynchronous protocol, reporting that the actions related to the TryAccess have been performed correctly;

- **Decision Combiner** (DC): this component has been added as result of the maturation process of the Continuous Authorization Engine component, in order to address the C3ISP pilots use cases. In particular, this component is invoked by the Multi-Resource Handler which sends it the results of the decision processes performed on each CTI of a given request. The main task of this component is to determine the set of CTIs on which the analytics will be executed in such a way that the policies of all these CTIs are respected and an *objective function* is maximized. A simple example of objective function to be maximized could be the number of CTI involved in the analytics. At M24 a very simple objective function is provided. Please refer to deliverable D8.2 for further details;
- **Multi Session Manager (MSM):** this component has been added as result of the maturation process of the Continuous Authorization Engine component, in order to address the C3ISP pilots use cases. The main task of this component is to keep track of a set of data to connect the usage session of each single CTI (which is managed by the original SM) with the multi resource access request it belongs to;
- **Context Handler** (CH): is the original entry point of the Continuous Authorization Engine component, and it coordinates the internal modules for the execution of the policy evaluation process. Due to the maturation process, this component now manages communication protocol for interacting with the **MRC**, using a subset of the usage control actions: *tryaccess, permitaccess, denyaccess, revokeaccess*, and *endaccess*. Moreover, new signatures have been added for the *tryaccess,* and *endaccess* interface in order to pass additional information required for the management of the multi resource access requests. The CH offers the following APIs:
  - **EndAccess**: invoked when a resource is released. It removes all records of the ended session and triggers post obligations;
  - **EndAccessResponse**: invoked as part of the asynchronous communication between the CH and MRH. Reports if the session has been closed correctly with correct execution of the post obligations;
  - **OnGoing**: used when the Continuous Authorization Engine is distributed. It sends remotely an evaluation request, to be evaluated by a remote instance of the Continuous Authorization Engine;

  - **OnGoingResponse**: sends the evaluation results for a request which has been computed remotely;
  - **RetrieveRemote**: invoked when there are multiple instances of the DSA Adapter and a request should be handled by a Continuous Authorization Engine which is remote with respect to the calling Event Handler;
  - **RetrieveRemoteResponse**: invoked from the remote DSA Adapter as response to a RetrieveRemote invocation;
  - **startAccess**: invoked by the MRH when the access to a resource is actually started. Triggers evaluation of the ongoing policy;
  - **startAccessResponse**: confirms the execution of all operations related to the startAccess invocation;
  - **tryAccess**: invoked by the MRH to ask the right to access a resource. Triggers policy vs request evaluation;
  - **tryAccessResponse**: invoked as part of the asynchronous protocol, reporting that the actions related to the TryAccess have been performed correctly;

- **Session Manager** (SM): is the components responsible for keeping track of the accesses that are currently in progress, in order to allow the continuous authorization phase;
- **Policy Decision Point** (PDP) is the component which evaluates security policies and produces the access decision. No modifications with respect to the original version of this components have been done;
- **Attribute Managers** (Ams) are the modules which manage the attributes required to evaluate the usage control policies. Each AM provides a custom interface to retrieve (and to update) the current values of the attributes it manages. The M24 version of the C3ISP framework integrates the following Ams:
  - **Identity Manager**: the Common Security Services subcomponent of the C3ISP framework offers an Identity Manager service which, in turns, provides a LDAP service. This service provides information about the C3ISP users, such as the organization they work for, the projects they are involved in, and so on. This information are exploited in the usage control policies to define the usage rights on the CTIs, and hence they must be retrieved in order to enforce such policies and make usage decisions concerning CTIs;
  - **MySQL**: the MySQL is a generic attribute manager suitable to be used by a large set of policies. The database can contain, for example metadata related to stored data, such as number of accesses, size, owner, etc., or related to entities accessing data. Attributes stored in MySQL database can be mutable, hence for these attributes it is necessary to implement PIPs able to periodically query these mutable attributes;
- **Policy Information Points** (PIPs) are interfaces for interacting with Attribute Managers in order to perform the main operations on attributes: *retrieve, subscribe/unsubscribe* and *update*. We developed the following PIPs in order to interact with the Ams listed in the previous point:
  - **LDAP**: the LDAP PIP is exploited to retrieve user attributes which are stored inside the Identity Manager. As discussed, the Identity Manager is implemented through an LDAP service to store basic information related to users (prosumers) willing to access or publish data stored in the DPOS. This PIP is in charge of

retrieving user attributes such as department, role, mail, etc. The PIP is highly customizable in order to deal with different structures of LDAP Identity Managers and to retrieve only attributes required by the policy;

- **MYSQL**: the MySQL PIP is dual to the formerly defined Attribute Manager. The MySQL PIP is designed to be general and easily adapt to any database defined with the MySQL software. The PIP exploits the Hibernate[13] framework to interface the Java code of the PIP itself with the database. The PIP is completely configurable via a configuration file, where the system administrator specifies username and password for the AM database, table(s) structure and format of the attributes.

5.1.3.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the Continuous Authorization Engine:

Figure 18: Continuous Authorization Engine API from Swagger UI

This is the ones for the Multi Resource Handler:

Figure 19: Multi Resource Handler API from Swagger UI

5.1.4. Obligation Engine

The **Obligation Engine** is a module that is responsible for the execution of specific operations when certain conditions take place. Such operations are *Usage Control Obligations* that are prescribed by the security policy (i.e. the sticky policy) associated to a specific data.

[13] http://hibernate.org/

Its architecture is not dramatically changed from that described in Deliverable D7.2 and depicted as follows. The only notable modification is the role of the private (internal) Event Handler instance, now with a mediator role assigned for all Obligation Engine modules. Also, the set of triggers and actions, regulated respectively by the Trigger and Action Engine have been explicitly represented. Both engines have methods to register new managed elements and to invoke them if requested to do so by the Obligation Engine Front End.

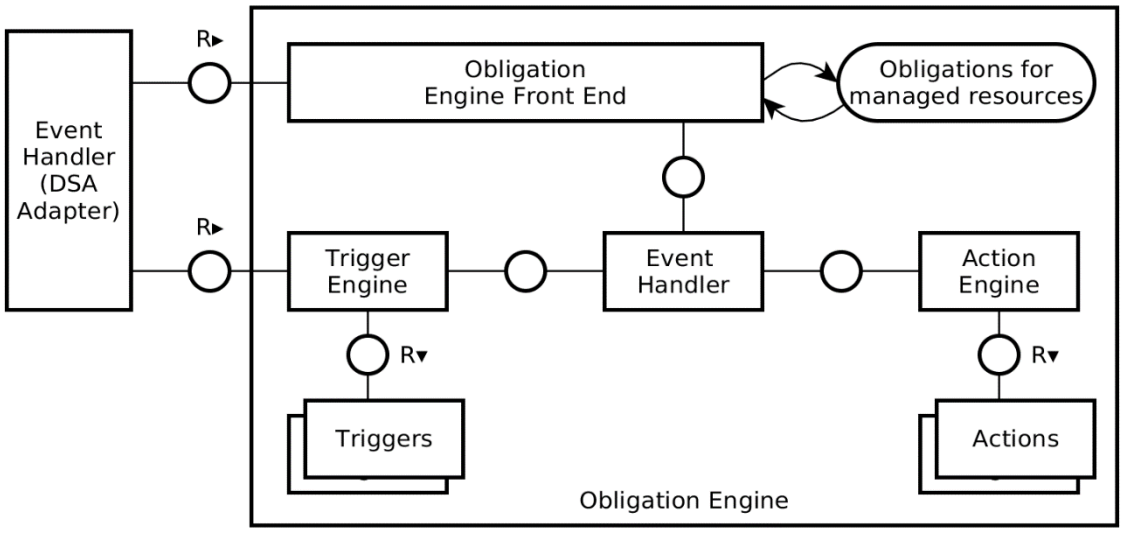


Figure 20: Obligation Engine architecture

Architecture Harmonisation with DMO Engine and WP4 Gateway's Orchestrator

In an effort to consolidate and simplify component design, development and evolution, three components and namely the Obligation Engine, the DMO Engine and the WP4 Gateway's Orchestrator (described in Deliverable D4.3) have been scrutinized. It emerged that the core functionalities of the Obligation Engine could be used as the foundation for the other two, organising the development of the components in way similar to a software product line. The respective sections of these components detail further the application of this concept.

5.1.4.1. Implementation and Integration Status

At M24 the Obligation Engine is implemented in its main components and it has been updated to meet M24 milestone. A number of new Actions (e.g. all the DMOs related to the Anonymization Toolbox, see D8.3) and Triggers (e.g. TriggerNewAnalyticsResultReady, see D8.2) have been added and more will be, in order to fully support all use cases exposed by the C3ISP pilots.

The described architecture is to be considered in its final version, even if refinements to some of the exposed functionalities may still occur, and notably with respect to supported actions and triggers, if it will be deemed necessary.

The major goal for the final prototype is to implement all needed triggers and actions; no updates are foreseen with respect to architecture and design.

5.1.4.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the Obligation Engine (in the screenshot, "ObligationInterface" as part of the Trigger Engine):

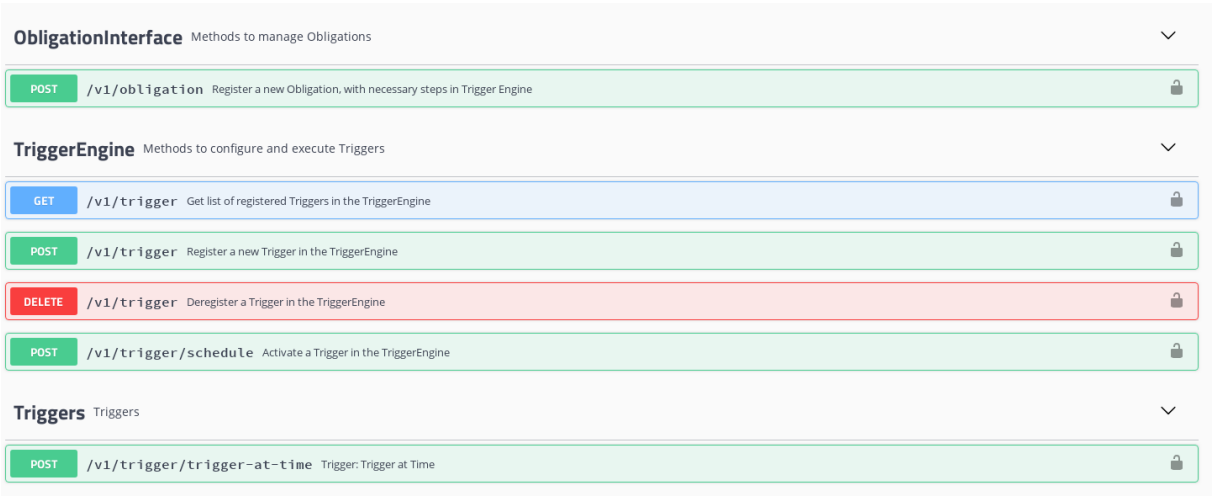


Figure 21: Obligation Engine API from Swagger UI

5.1.5. DMO Engine

The **Data Manipulation Operation (DMO) Engine** is the component in charge of executing the Data Manipulation Operation returned as a result of the decision process on the data and/or by any obligation as prescribed by the DSA. In fact, besides determining whether the data can be accessed or not by the requestor, the decision process also determines a set of operations that must be executed on such data before being released to the requestor. As an example, a DSA paired to a system log could require that all the IP addresses present in such log must be anonymized before releasing this log to a third party. A similar action may be mandated also in case a retention period for the log is expired, irrespective of any access request. The DMO Engine is the component of the DSA Adapter devoted to perform such anonymization operation on the log (i.e. by interacting with the external Anonymization Toolbox service, see D8.2).

Architecture Harmonisation with Obligation Engine and WP4 Gateway's Orchestrator

The architecture of the DMO Engine has been refactored, in an effort to streamline the design of the 3 similar C3ISP components (Obligation Engine, DMO Engine and Orchestrator in WP4 Gateway, see deliverable D4.3 for the latter) and to move towards approaches resembling those studied for the software product lines [9].

As a result, the architecture of the three components has been harmonized, after having refined the initial design and use cases. However, the implementation of the three components is slightly different, where specific functionalities are rendered on top of a common core. The implementation of each component is detailed in Deliverable D8.2, while its architecture is depicted as follows.

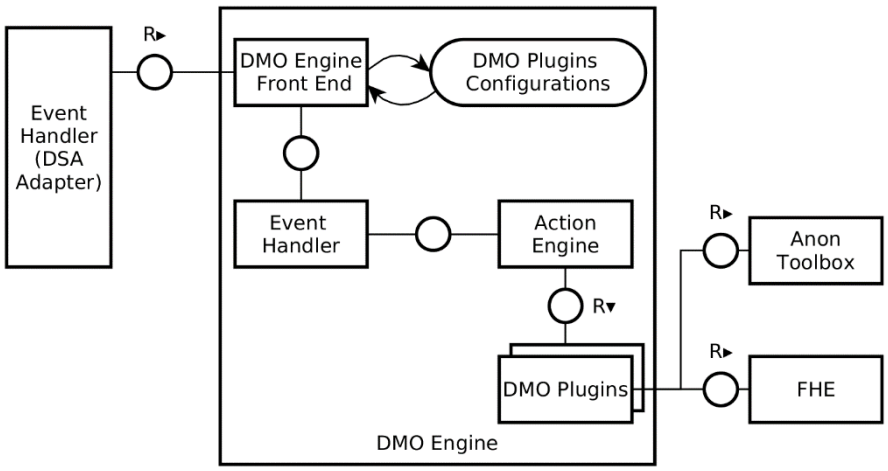


Figure 22: Data Manipulation Engine architecture

The Obligation Engine components that are used in the DMO Engine are:

- The Event Handler;
- The Action Engine;
- The conceptual foundation of the Front End (even though the exposed API is different);
- The same persistency management component.

The Trigger Engine is not deemed helpful for the DMO Engine. Specific actions are implemented in order to support the integration with the Anonymization Toolbox (see D8.2) and the Full Homomorphic Encryption (see Sections 8.2) components.

The described architecture is to be considered in its final version, even if refinements to some of the exposed functionalities may still occur, and notably with respect to supported actions (e.g. support for the FHE), if it will be deemed necessary.

5.1.5.1. Implementation and Integration Status

DMO Engine is integrated with the Anonymization Toolbox with respect to a subset of its functionalities, while for FHE component, integration is expected for M28.

The major goal for the final prototype is to achieve a full and complete integration with the data manipulation operations.

5.1.5.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the DMO Engine:

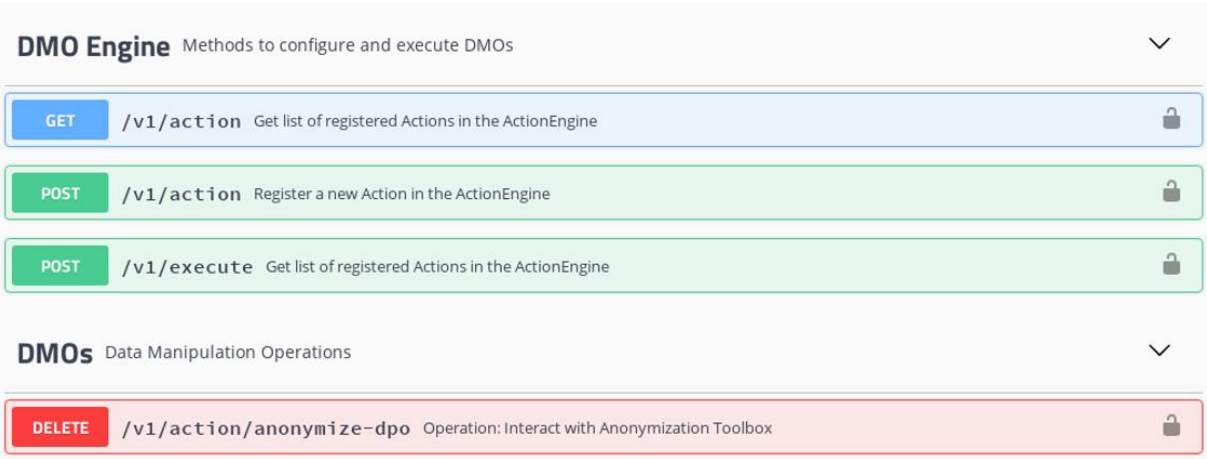


Figure 23: DMO Engine API from Swagger UI

5.1.6. Bundle Manager

At M24, the **Bundle Manager (BM)** aims at creating, deleting and retrieving packets of data by invoking specific functions in **DPOS** component (*Data Protected Object Store*). The DPO packet is a bundle of four files, in which the cyber-threat information data file first is encrypted with AES cryptosystem and then sent to DPOS component for storage.

The four files to store are:

1. The cyber-threat information data, called CTI file (or CTI data);
2. The metadata file, which is a description of the packet;
3. The DSA policy file retrieved from the DSA Manager subsystem;
4. The hash (or hash code) for all the previous files, to guarantee data integrity.

The DPO bundles can be retrieved or be deleted via BM API. To complete these actions, the BM API interacts with the following components:

- Key & Encryption Manager API (part of the CSS subsystem), which permits (a) to generate, renew, store cryptographic keys and (b) to encrypt, decrypt data;
- DPOS API (part of the ISI subsystem), which is the API of the *Data Protected Object Storage* component;
- DSA Manager Gateway (part of the DSA Manager subsystem), which exposes the API used to retrieve DSA policy (in UPOL representation).

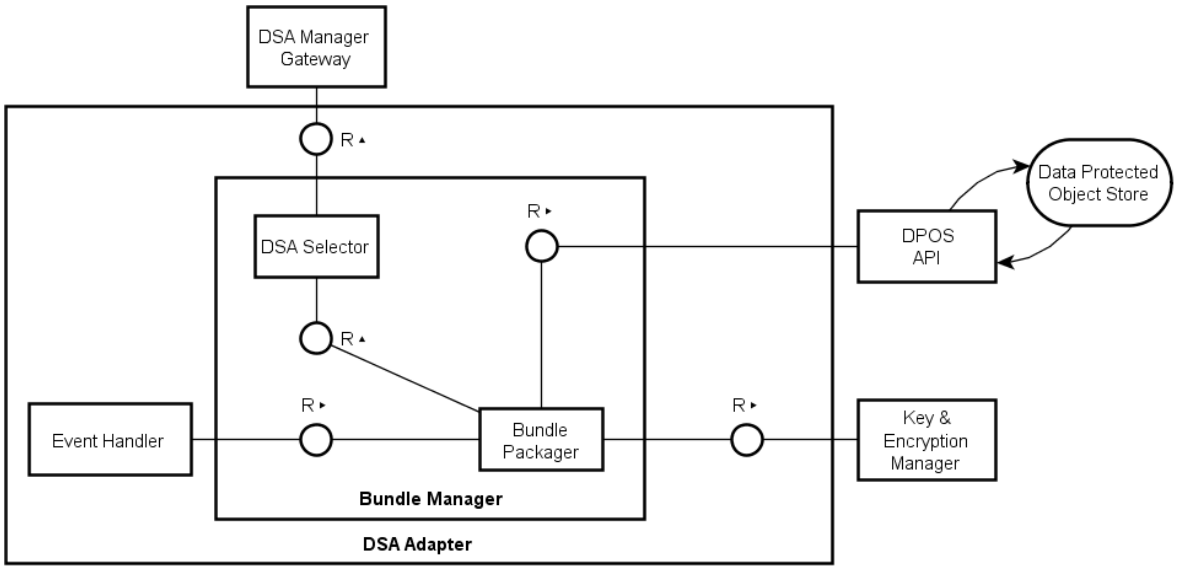


Figure 24: Bundle Manager architecture

5.1.6.1. Implementation and Integration Status

At M24 the Bundle Manager APIs are stable, up and running.

The interfaces permit:

- **To create a bundle,** the Bundle Manager has to subscribe for a "bmc" event from the Event Handler with a POST request. This request should contain the DSA-Id, the metadata file and the cyber-threat information (CTI) file. If the creation of a new bundle is successful (i.e. stored in the DPOS), then the result is a DPO identifier (DPO-Id).

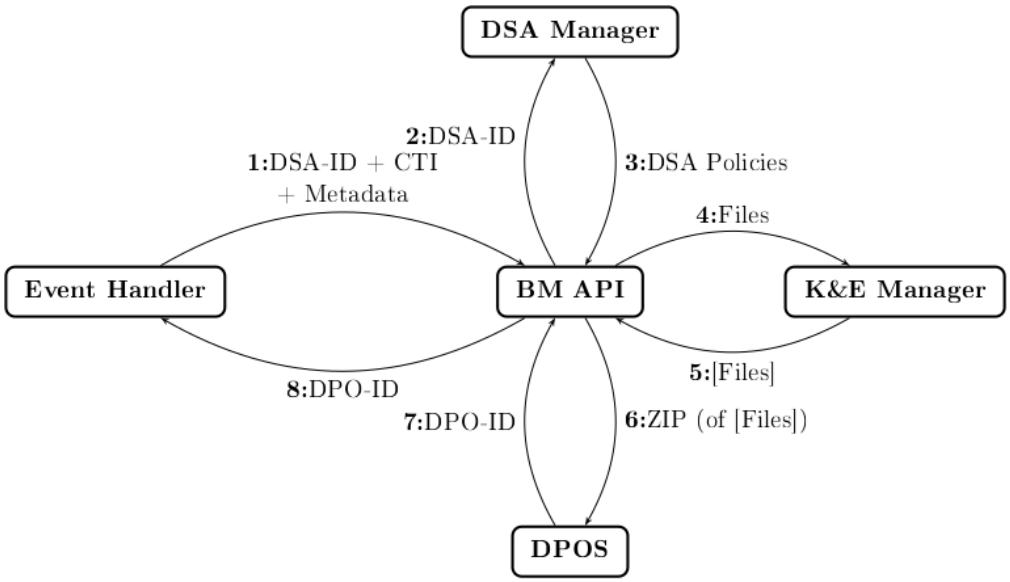


Figure 25: Create a DPO process. The symbol [Files] denotes the data is in encrypted form.

In the bundle data, Metadata file is not encrypted, the DSA policy file is encrypted with the key identified by the DSA-Id and the CTI file is encrypted with IAI-C3ISP-ID as the identifier for the key. Note that the identifier IAI-C3ISP-ID is used for analysis purposes in IAI platform.

For encrypting the CTI file, we can use two encryption systems. In the case where the CTI file is used for analytics services that work on data in clear-text, then only AES is taken into account. On the another hand, if the CTI file is used for FHE analysis, then some specific fields in the file which contains confidential data (e.g. IPv4 in the case of FHE analysis) are firstly encrypted with Kreyvium homomorphic-friendly cryptosystem (see Section 8.2), secondly the CTI file is entirely encrypted with AES (double encryption).

- **To retrieve a DPO**, we mostly follow the same steps as those of creating the bundle.

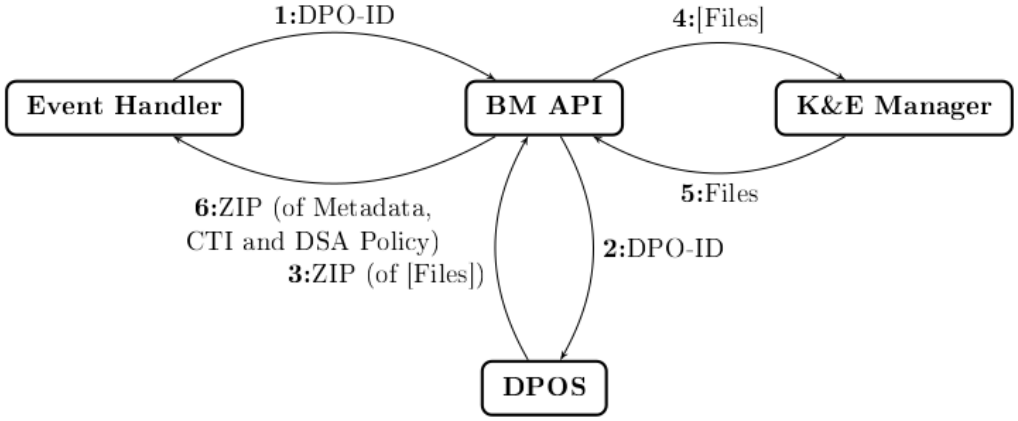


Figure 26: Retrieve a DPO. [Files] denotes an encryption of Files

When the BM receives a "bmr" event from the Event Handler for retrieving a DPO bundle, the BM proceeds invoking the read DPO function from the DPOS, then decrypts the CTI file by means of K&E Manager (thanks to the DSA ID stored in the Metadata file), finally creates a zip file for these four files and sends it back to the Event Handler. Note that between steps 5 and 6, the API performs a hash check to ensure the integrity of the data, but the verification of data integrity will be improved in the next version of C3ISP Framework.

- **To delete a DPO**. It is fairly straightforward from the point of view of the BM. When receiving the "bmd" event from the Event Handler, the BM invokes the corresponding DPOS API for the deletion of the given DPO-Id.

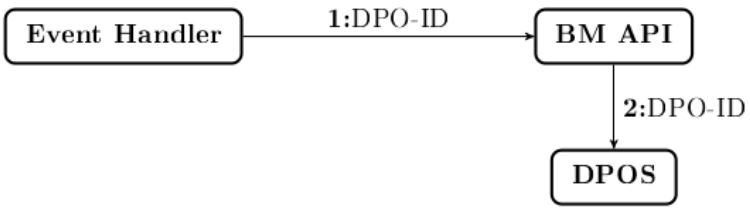


Figure 27: Delete a DPO

The major goal for the final prototype is to integrate the BM with the K&E Manager component in such a way to prepare FHE encrypted data in advance for FHE analysis later.

5.1.6.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the Bundle Manager:

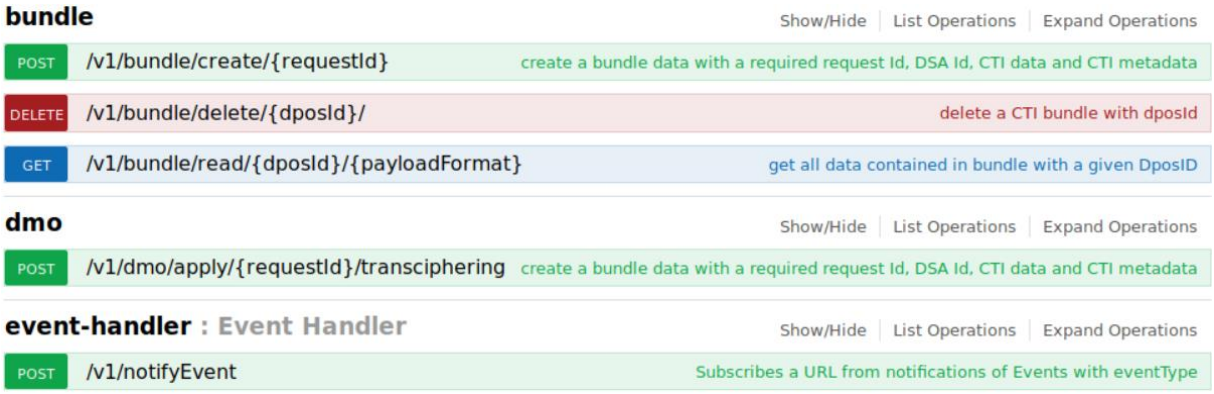


Figure 28: Bundle Manager API from Swagger UI

5.2. Format Adapter

The **Format Adapter** is the component of the C3ISP Framework in charge of the following functions:

- **Convert (/convert API)**: adapts the format of input CTI data to a standard format (STIX) to be easily processed by the various C3ISP components. To accomplish this, the Format Adapter is called by the ISI API;
- **Convert for Data Lake (/convertDL API)**: formats the CTI data appropriately for a given analytics service before executing it. This is realised when the Buffer Manager calls the Format Adapter in order to have the data prepared in the right format for the analytics.

5.2.1. Implementation and Integration Status

At M24 the Format Adapter is implemented as a RESTful web service implemented in NodeJS deployed in the Information Sharing Infrastructure (ISI).

The integration status of each CTI data can be found in the table below. The table lists the formats interpreted as input by the Format Adapter, which are converted into STIX when the data is received by the **convert** API (convert API is called by the ISI API during the createDPO):

Table 33 – Released software components for C3ISP Framework at M24

CTI data	Integration Status
Monitoring of connections to malicious hosts	✓
Monitoring of Domain Generation Algorithm DNS-request	✓
Email Analysis	✓
Firewall Schema	✓
Anti-Malware Schema	✓
Security Report Sharing	✓
Enterprise Pilot (Intrusion Detection Events, Malware Events, Network Traffic Events, Web Events)	×

Currently **convertDL** API simply removes the STIX envelope before: this API is called by the Buffer Manager when creating the Data Lake where the analytics will run.

The major goals for the final prototype are to make the Format Adapter able to automatically detect the format on input (CTI data) and translate it into the required standard format. Further, the convertDL API will require to convert data to specific formats required by the analytics; the list of this formats include CSV (Comma Separated Value) and CEF (Common Event Format).

5.2.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the Format Adapter:

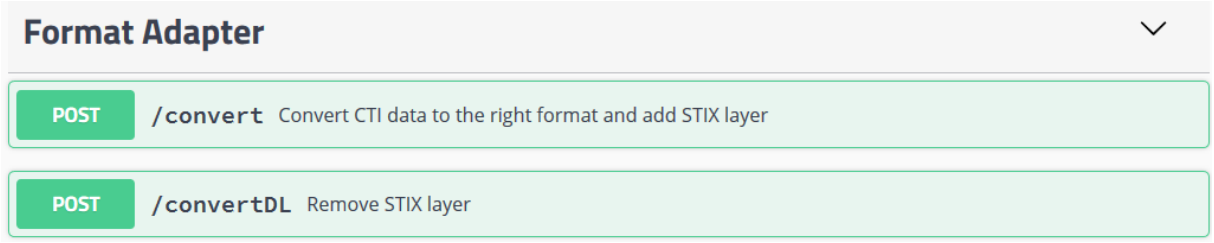


Figure 29: Format Adapter from Swagger UI

5.3. Buffer Manager

The **Buffer Manager** is the component of the C3ISP Framework that manages Data Lakes (buffers). A Data Lake is a component that is used to store data before and after analytics tasks executed by C3ISP-aware analytics services or Legacy Analytics services. Data Lakes are of two types: **Data Lake Buffer** (DLB) and **Virtual Data Lake** (VDL) that are different in terms of how they access and read the data. They are transient, meaning that they are created and reserved only for the execution of a specific analytics service; then, they get removed.

In the current design, the Buffer Manager supports three implementations of Data Lake that store data in different fashions:

- On a local file system: this is the simplest and mainly used for testing;
- In a MySQL instance: this is used for example by the BT SATURN tool;
- On a distributed file system (i.e. Hadoop File System – HDFS): this is the most advanced and enables the execution of big data analytics services.

The component is designed and implemented to allow the creation of new Data Lakes (for different storage types) in the future, would it be necessary. The current Data Lakes support satisfies the analytics service requirements.

The Buffer Manager implements these functionalities:

- Create instances of a Data Lake, i.e. provide a VDL and a DLB implementation for each storage type. The return value is a URI that will be used by the analytics services to read and, optionally, write CTI data (write access is needed by those analytics that stores their result temporarily);
- Access an instance of a Data Lake using the URI and store data on it;
- Fetch DPOs from the DPOS on the (central) ISI using the ISI API;
- Delete an instance of a Data Lake using the URI.

5.3.1. Implementation and Integration Status

At M24 the Buffer Manager implements the following features for managing two different implementations of Data Lake:

- A **File System Data Lake** that uses a traditional file system to store data for the Analytics. Data Lakes are created in a dedicated folder on the node the Buffer Manager is running (e.g. ISI VM). This Data Lake can be exported to other nodes using third party mechanisms (e.g. NFS). URIs follow this model:

  `file:///opt/isi/datalakebuffer/<data_lake_name>`

  Segregation between different <data_lake_names> is performed via regular Unix filesystem permissions;

- A **MySQL Data Lake**, built for analytics services that need to read data from a relational DBMS. This implementation provides a MySQL URL. Credentials are provided as properties in the URL in order to grant access only to the particular Data Lake instance:

  `jdbc:mysql://iaic3isp.iit.cnr.it:3306/<data_lake_name>?usr=<user>&psw=<password>`

  Segregation between different <data_lake_names> is done by instantiating different user accounts per each analytics execution.

The Buffer Manager is implemented via a RESTful web service that provide the following API:

- A **prepareData** endpoint that creates a Data Lake instance and populates it with data read from the ISI API. Returns the instance's URI;
- A **prepareEmptyDataLake** endpoint that creates an empty Data Lake instance and returns the URI;
- A **populateDataLake** endpoint that writes data inside an existing Data Lake. This function does not read the data from the ISI: it populates the Data Lake with the content provided with the API call. This API is only used by the DMO Engine component to safely store DPOs that have not been manipulated yet, protecting them from unauthorized access;
- A **releaseDataLake** endpoint that can be used to delete any Data Lake by providing its URI and the access credentials.

The flow about how to use these API is described in detail in Section 9.4.

The current version of the Buffer Manager is integrated with the ISI API, which are used to populate the Data Lakes it creates. This is enough to have a working end-to-end workflow for the execution of analytics services, but it makes some assumptions regarding the data formats. In fact, the final prototype will also include:

- Integration of the C3ISP **Format Adapter** component, which will convert data as required by the different Analytics services;
- A new Data Lake implementation for the **Apache Hadoop Distributed File System (HDFS)**, with the goal of integrating Hadoop big-data analysis tools into C3ISP Analytics services.

5.3.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the Buffer Manager:

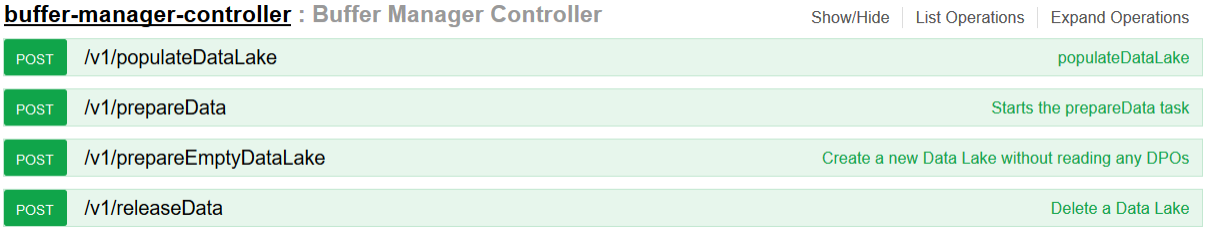


Figure 30: Buffer Manager from Swagger UI

5.4. Data Protected Object Store API

The **Data Protected Object (DPO) Store API** is the external interface of the DPO Store, and it provides functions for managing the storage of DPOs. The DPO Store API is used by the ISI components to manage the storage of Data Protected objects. It is also indirectly used by the IAI subsystem to search and retrieve DPOs for use in collaborative analytics, via the ISI API.

The DPOS prototype supports the following functionality:

- **CreateDPO**: Given a CTI file, and the associated DSA file, hash code and a DPO metadata header, create a DPO in the DPO Store;
- **ReadDPO**: Retrieve the four components of a DPO from the DPO Store repository, given its DPO ID;
- **DeleteDPO**: Given a DPO ID, delete the corresponding DPO from the repository;
- **SearchDPO**: Given a JSON-based search string (described in detail in D8.2), query the DPO metadata repository, and return a set of metadata entries corresponding to the matching DPOs. This method returns either a set of DPO IDs or a set of full DPO metadata entries, based on the Boolean *longResultFlag* parameter.

The main architectural refinement introduced since D7.2 has been the design and implementation of a search function. To enable this search, the DPO header provided by the DPOS client at CreateDPO now contains a set of searchable DPO metadata. The query schema used by SearchDPO allows the DPOS client to search and filter DPOs based on information contained in their metadata headers.

5.4.1. Implementation and Integration Status

At M24, the prototype is fully functional. In addition to the storage functionality described in the design phase (see D7.2), the DPOS additionally supports a search feature, which allows search on a set of metadata fields stored as part of the DPOS. Both the metadata fields and the query languages use a JSON-based format (see D8.2).

5.4.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the DPOS API:

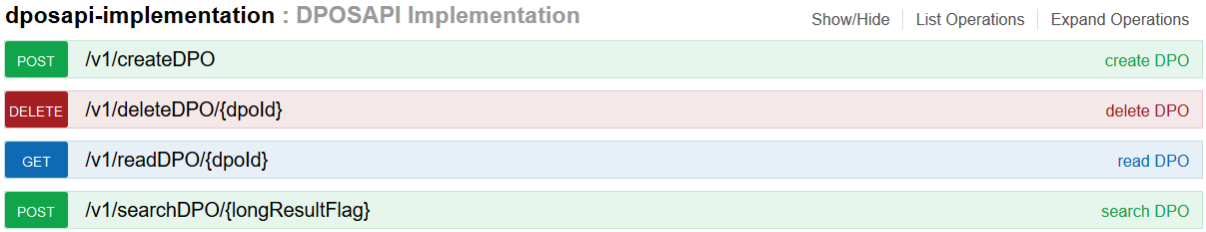


Figure 31: DPOS API from Swagger UI

5.5. *Data Protected Object Store*

The **Data Protected Object Store (DPOS)** persistently stores the CTI data provided by the Prosumers in the form of the C3ISP data bundle (or DPO – Data Protected Object), i.e. the CTI data, the corresponding DSA, hash code, and DPO metadata. It functions as the backend to the DPOS API.

5.5.1. Implementation and Integration Status

The DPO Store has been implemented as two separate repositories: the ***search backend***, and the ***storage backend***. Both repositories are implemented using in-house software, which connects the DPOS API (see Section 5.4) to the off-the-shelf products acting as the repositories:

- The ***search backend*** uses *MongoDB*[14], a document-oriented NOSQL database. It stores and searches DPO metadata, thus enabling C3ISP users and applications to search and filter data-protected objects;
- The ***storage backend*** uses *Apache Hadoop Distributed File System (HDFS)*, the filesystem component of the Apache Hadoop[15] suite of open-source data-processing tools. It is used by the DPO Store purely as a large-scale distributed filesystem, which stores the remaining three DPO components: the CTI data file, the associated DSA file, and a hash signature. Additionally, the DPOS can be configured to use the *local filesystem* instead of HDFS. This configuration is meant for use in a small-scale deployment, as in a Local ISI.

The DPOS in-house software connects the DPOS REST API with the search and storage back-ends. It abstracts the backend implementation from the DPOS API client, and allows the choice backend options (as in the case of HDFS vs local-filesystem storage implementation), depending on the needs of the target environment.

The main architectural refinement introduced since D7.2 has been the splitting of the DPOS into two repositories: the search backend, and the storage backend, to facilitate the search functionality introduced in the DPOS API.

Furthermore, we have provided two alternative implementations of the storage backend: a large-scale distributed filesystem, for use within a central ISI deployment, and a lightweight local-filesystem option, for use in a local ISI deployment, or for testing.

[14] https://www.mongodb.com/

[15] http://hadoop.apache.org/

5.6. ISI API

The **ISI API** is the front-end component providing services to the C3ISP actors (Prosumers or Prosumers applications, or the IAI subsystem). It orchestrates the processing flow with the other ISI components, interacting with the DSA Adapter or the Format Adapter (see the data flow diagrams in section 9). All the APIs are actions that are subject to the (DSA) policies associated to each CTI data object (i.e. to each DPO).

The list of the operations of the ISI API are the one reported in D7.2, but we have renamed from "*Action* CTI" to "*Action* DPO" such as Create DPO, Read DPO, Move DPO, Delete DPO (Prepare Data is also another available API). This makes clear that we are dealing with Data Protected Objects (DPO); in fact, the data these functions receive in entry is already a CTI and using the term DPO is more appropriate and clear. Further there is a new API that is in used to search for DPOs matching some conditions; this API is a proxy for the search DPO functionality exposed by DPOS API (see Section 5.4 for its description).

5.6.1. Implementation and Integration Status

At M24 the ISI API is implemented and integrated with the DSA Adapter Front End and with the DPOS API for what concerns the implementation of the search functionalities.

The major goals for the final prototype are to tighten the integration with the CSS (for example, with respect to user authentication) with no other expected updates.

5.6.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the ISI API:

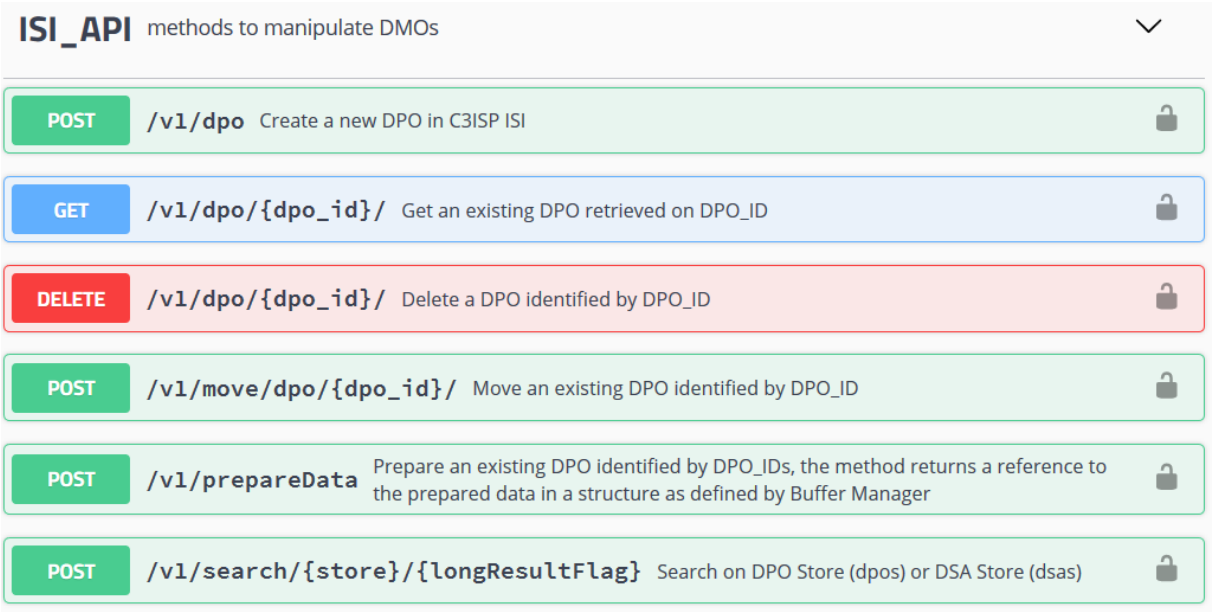


Figure 32: ISI API from Swagger UI

6. Subsystem Updates & Status: IAI – Information Analytics Infrastructure

The Information Analytics Infrastructure (IAI) allows Prosumers to request the execution of analytics services on the data protected and shared by the ISI. It supports both so called C3ISP-aware analytics services, jobs that can exploit the full capabilities of the C3ISP Framework, and so called legacy analytics services (i.e. already existent analytics), that can run on the shared data but have limitations.

This section reports on the revised IAI that has been developed at M24 and deployed into the C3ISP Test Bed.

The IAI is made up of the following components, described in detail in the next sections:

- C3ISP Analytics Engine: to run data analytics jobs that exploit the full power of the C3ISP Framework;
- Service Usage Control Adapter: to protect the Prosumers' usage of the analytics services;
- Legacy Analytics Engine: to provide the interface for using a legacy analytics engine;
- Virtual Data Lake: to implement a "transient" or "per-call" data lake used for analytics processing by the legacy engine;
- IAI API: to provide the interfaces for external interaction with the Prosumers (or their applications) and other C3ISP subsystems.

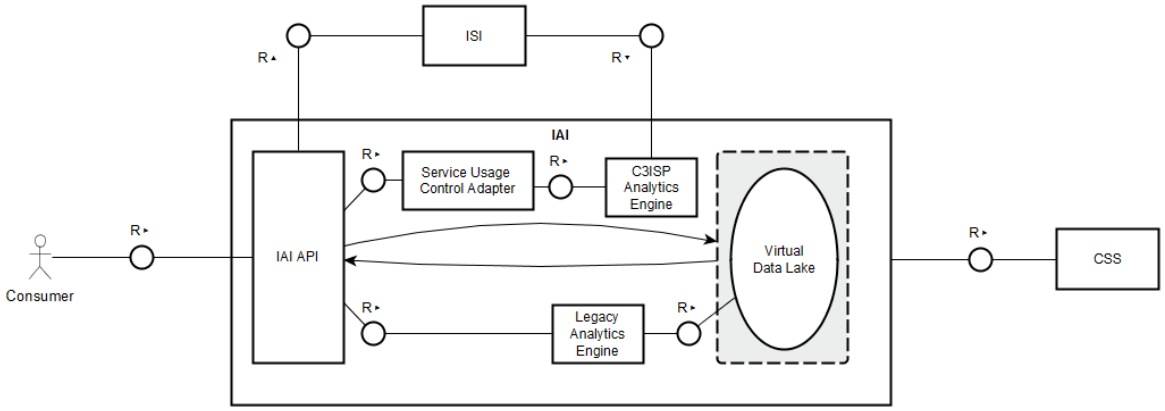


Figure 33: Information Analytics Infrastructure

The IAI interacts with external clients:

- the Consumer (or an application on behalf of it), which would use the IAI API to execute analytics jobs on the data shared through the ISI;
- the Information Sharing Infrastructure (ISI) to request the data for processing, subject to DSA policies;
- the Common Security Services (CSS) for secure auditing of its activities and for identity management.

6.1. C3ISP Analytics Engine

The **C3ISP Analytics Engine** is a set of methods and tools offered by the C3ISP Framework to extract additional knowledge from information shared by Prosumers. The offered analysis tools include *computational intelligence functions,* in particular clustering and classification algorithms, data aggregation and correlation functions, statistical analysis tools, and data visualization primitives. These functions are either implemented through open source libraries for Machine Learning (ML) and statistical analysis, in particular WEKA[16] and Scikit-Learn[17] libraries, and tools for Big Data analysis.

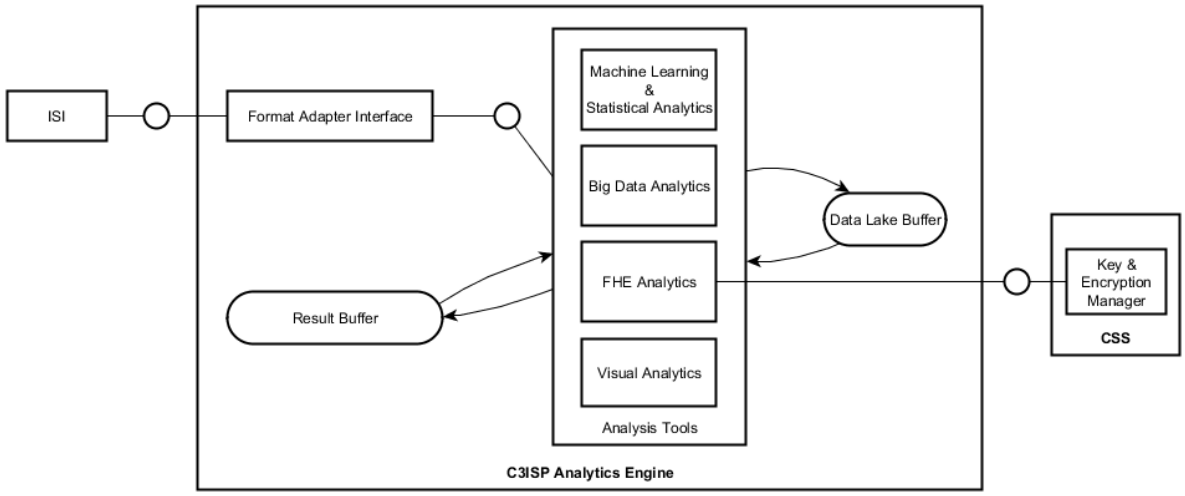


Figure 34: C3ISP Analytics Engine

The infrastructure depicted in Figure 34 relies on a core of **Analysis Tools** and on three functional modules to handle the data flow in the engine. The **Format Adapter Interface**, is an interface to the Format Adapter component (via the ISI API) already described in the previous section, which will prepare the format of information, from the structured CTI format, to the one needed by the required analysis algorithm. The **Data Lake Buffer** is a temporary storage in which pieces of information used for analysis are stored (see also Buffer Manager in Section 5.3). The specific structure for storage in the buffer will depend on the specific analysis to be performed, i.e. it could be a simple data buffer to store temporarily the actual parameter of the analysis function, or it can embody a structured or unstructured database for big data storage, to be used as buffer for the map-reduce operations. The **Result Buffer** will contain temporary and final results, acting thus both as a complimentary component to the Data Lake Buffer, and to store the final results before they are sent to the ISI via the Format Adapter Interface.

All the analytics functions considered are compatible with the Data Manipulation Operation described in the ISI and their analysis is completely under the control of the C3ISP Framework.

The following subsections describe the status of implementation for this First Version of the C3ISP platform (M24).

6.1.1. Implementation and Integration Status

The analytics engine is currently integrated in the C3ISP operative workflow and an instance of it is currently running on a dedicated machine in the CNR premises. At M24 the C3ISP

[16] http://www.cs.waikato.ac.nz/ml/weka/

[17] http://scikit-learn.org/stable/

Analytics Engine integrates a first set of analytics offered to the various pilots through the IAI API. The functionalities of the available analytics have already been tested, also verifying the correct reception of data from the ISI API with applied DMO when needed, data conversion operated by the Format Adapter, data analysis and delivery of results to requesting prosumer.

The major goals for the final prototype are the integration of a larger set of analytics needed to satisfy all the pilot requirements related to expected results, and an efficient management of very large sets of data (Big Data). Both of these activities are currently being carried on as part of the maturation activities.

6.1.2. FHE Analytics

At M24 the FHE Analytics are used to operate treatments on encrypted words and focus on encrypted IPv4 addresses which have two advantages for homomorphic computations, they are short (32-bits) and fixed-sized words.

FHE Analytics offer tools which permit to detect if an encrypted IPv4 belongs to a set of encrypted IPv4s.

6.1.2.1. Implementation and Integration Status

As a reminder, the backend employs the compilation chain Cingulata developed by CEA [10]. This tool permits to operate homomorphic cryptography on binary data. That is, it permits to write a user-friendly C++ program to define computations on encrypted data (ciphertexts). This program permits to construct a Boolean circuit of BLIF type which is used for two tasks:

1) Parameter generation for Brakerski/Fan-Vercauteren homomorphic cryptosystem [11][12] (named B/FV or FV in the literature) stored in a xml file;
2) (Homomorphic) Evaluation, that is to execute FHE Analytics.

Let us discuss about modifications brought to FHE Analytics implementation:

- IPv4 representation has been modified in the C++ program to decrease the number of AND gates. This impacts time performance as shown in D8.2, Section 7.2.4. It is represented by one 32-bit integer rather than four 8-bit integers;
- Elementary data have fixed size but input/output data (e.g. list size) in a FHE Analytics is variable. On another side, the BLIF circuits have fixed-size inputs and outputs. Pre-computations and post-computations on clear data (plaintexts) have been done to allow FHE Analytics on variable-size data and to obtain variable-size results;
- Creation of BLIF circuit databases for each FHE Analytics and for lists' size smaller than 150. This saves time by avoiding BLIF generation at each call;
- Offline encryption of a toy IPv4 set/list with fixed cryptographic key set. It helps for benchmarks and is useful for use with real data.
- List location is not hardcoded anymore, it is now an optional argument given by Prosumer to enable to test membership in several lists.

The implementation of an FHE Analytics regarding blacklist checks is available and deployed in the C3ISP Test Bed (see API in Section 6.1.2.2).

The major integration step to do in the next version is to integrate the transciphering: it means that when receiving a request for analyzing a CTI bundle packet which has been encrypted by Kreyvium, BM should invoke K&E Manager for transcrypting and K&E Manager should store

these FHE encrypted data in order that FHE-analysis component can retrieve them before processing.

Another major goal is to improve time/memory/communication performance. We plan either to employ TFHE (Torus – Fully Homomorphic Encryption), the current most performant homomorphic cryptosystem (a collaborative work with CEA, best paper awards in Asiacrypt 2016) [16][17][18], or to modify data encoding and to choose to apply a batching technique [13][14][15]. In the both cases, we estimate that the execution time can be clearly improved at least 20 times up to 50 times. In terms of storage, we can obtain an important improvement: a some hundred Kb for one IP instead of 4 Mb for an IPv4.

On one hand, TFHE consists in multiple optimizations of the GSW cryptosystem [19][20][21]. It will be integrated into Cingulata in the next months. It allows to evaluate an arbitrary Boolean circuit composed of binary gates, over encrypted data, without revealing any information on the data. It is independent of the multiplicative depth parameter which restricts levelled homomorphic cryptosystem like B/FV. This is due to a very fast gate-by-gate bootstrapping. This operation permits to perform more complex treatments by decreasing the noise produced by homomorphic operations. If the noise is too important, it is not possible anymore to do homomorphic operations. In addition, time performance can be easily estimated by counting the number of binary gates. Each binary gate takes about 13 milliseconds (on a single-core machine). It offers better performance than B/FV. We estimate that combining it with transcription mechanism is a good strategy to improve performance on C3ISP Homomorphic Algorithms during Y3.

On the other hand, batching (also called packing or Single Instruction Multiple Data (SIMD)), enables to encrypt multiple messages in one ciphertext and thus:

- To evaluate the same homomorphic operations on each of these messages in parallel using the packed ciphertext;
- To decrease the ciphertext expansion that is the ratio between the ciphertext size and the plaintext size.

In our case, it consists in processing several bits at the same time rather than one. Each of these bits is encoded into a *slot*. After which, the same homomorphic operations is evaluated on each slot in parallel. The plaintext is represented by a polynomial with binary coefficient rather than by a single bit.

The first software release proposes two APIs named **belongsBlacklist** and **countAppearance** (see Section 6.1.2.2). They permit respectively to test if an IPv4 belongs to a blacklist (circuit c3isp-membership, with no duplicate, see D8.2, section 8.2.4) and to count the number of occurrences of an IPv4 in a blacklist (circuit c3isp-multiplicity, with possible duplicates). Both takes two arguments: the requested IPv4 and the considered list size x. Time and memory performance depends on the list size. In D8.2, Section 8.2.4, we give benchmarks for a list of size 100. Currently, one blacklist with approximately 600 pseudo-random IPv4s is considered. Blacklist name is hardcoded and the first x are considered during the test. It has to be modified in the next version to consider different blacklists. In addition, transcription is not yet integrated and has to be in next release.

6.1.2.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the FHE Analytics:

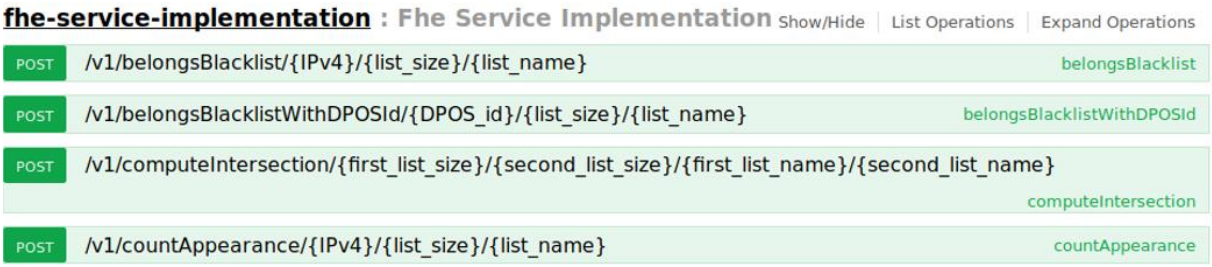


Figure 35: FHE Analytics API from Swagger UI

6.1.3. Interactive 3D Visualisation

Results from the C3ISP Analytics Engine are presented as an interactive 3D visualisation. The visualisation provides information on the number of attacks, number of susceptible nodes, number of infected nodes, contextual information, and shows how cyber-attacks propagate across the planet.

This component of the analytics engine is powered by 3D Repo's collaboration platform for 3D data, which can be accessed through a web browser or virtual reality (VR) headset. Through the interactive visualisation, areas of interest can be flagged for review, nodes can be filtered and grouped, and viewpoints can be saved.

6.1.3.1. Implementation and Integration Status

At M24, the 3D Visualisation component is in its third iteration following feedback from C3ISP project partners, Figure 36.

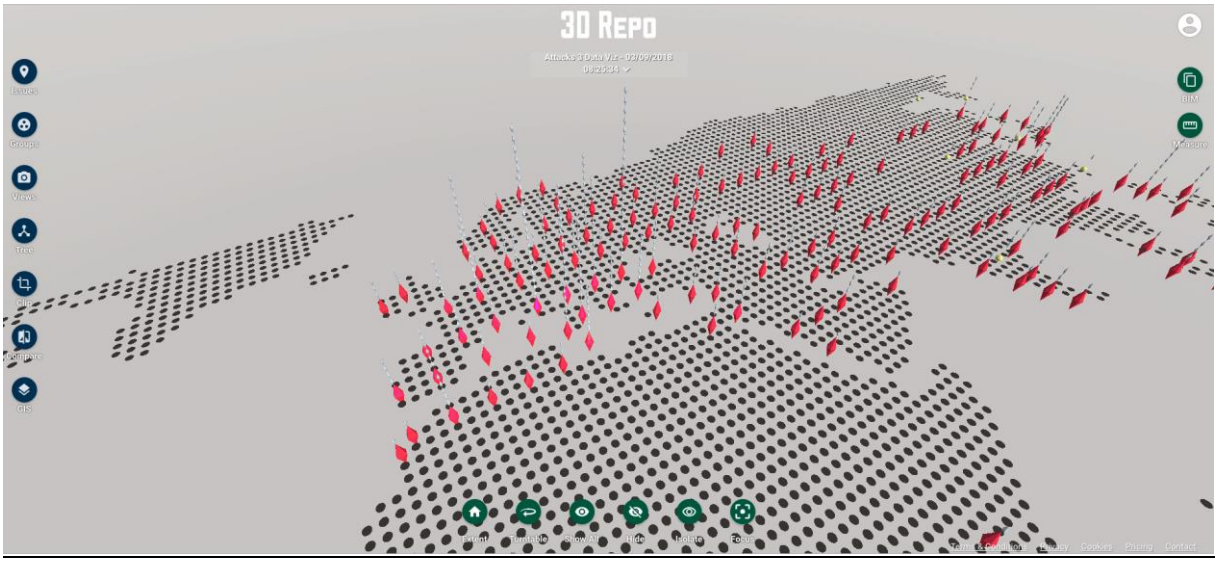
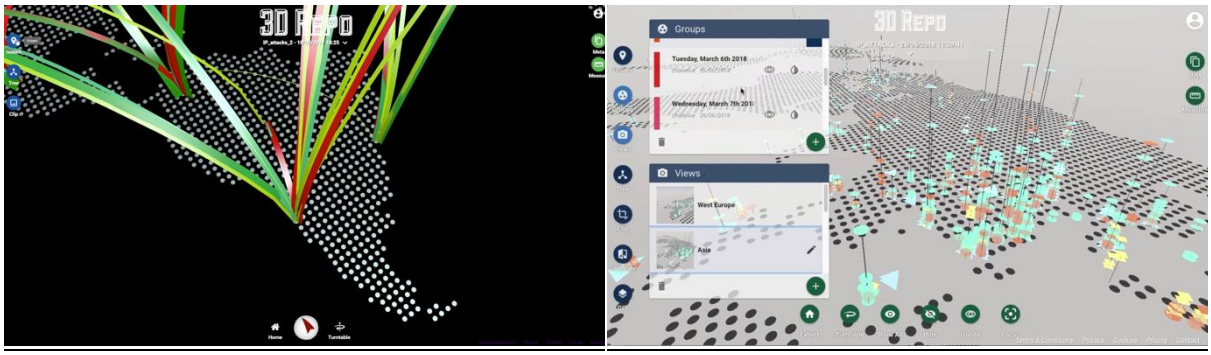


Figure 36: 3D visualisation of attack data (Version 3)

To visualise anonymised results from the analytics engine on the virtual map, attack data is grouped by geo-location. Information about the number of attacks, number of susceptible nodes, and number of infected nodes are grouped spatially into non-overlapping windows of size, *x*, for longitude and latitude. Noise can also be introduced to further anonymise results from the analytics engine.

In the first prototype, information such as attack vectors are shown in the visualisation, Figure 37(a). However, the visualisation becomes difficult to interpret as the number of data points displayed increases and when the data is anonymised. Figure 37(b) shows the second prototype, which introduced the ability to group nodes – e.g. attack data can be grouped by day – and save viewpoints within the visualisation.



(a) **(b)**

Figure 37: (a) Version 1 visualisation with attack vectors; (b) Version 2 visualisation with ungrouped attack data

The visualisation platform is implemented using Unity game engine[18], which is a popular cross-platform game engine with support for a wide range of desktop, mobile, and VR systems, Figure 38.

Figure 38: Interactive 3D visualisation on HTC's Vive VR headset

For the final prototype, the aim is for seamless integration of the interactive 3D visualisation through automation of tasks, such as grouping of nodes, and improved filtering and querying of the analytics results.

[18] https://unity3d.com/

6.2. Service Usage Control Adapter

The **Service Usage Control Adapter** is the component of the C3ISP architecture devoted to protecting the services offered by the IAI (described in the following section 6.5) from unauthorized accesses and usage. This adapter, similarly to the *DSA Adapter* previously described (Section 5.1), implements a Usage Control engine, thus being able to perform traditional access control along with continuous authorisation and obligation enforcement, which characterise the Usage Control model. Hence, the Usage Control policies enforced by this component define, for each of the services offered by the IAI, who can perform which analytics operations under which conditions, and whether these operations can be carried on over time. In this case, differently from DSA which is defined by who shares the data and it is paired with the data itself, the Usage Control policy is defined by the entity which provides the service, and it is paired with the service to be protected. For instance, since the homomorphic encryption based services are very resource-intensive, the provider of such a service could define a Usage Control policy which states that only two requests can be served at the same time. When the third request is received, the Usage Control policy checks the priority assigned to the incoming request against the priorities of the previous two (which are running) and if the former is greater, the running request with lower priority will be suspended to serve the third request. The suspended request will be resumed as soon as one of the other request will have been served.

The architecture of the *Service Usage Control Adapter* is the same as the architecture of the *DSA Adapter* (shown in Figure 10), since both Adapters enforce Usage Control policies expressed exploiting the same executable language. The differences between the DSA Adapter and the Service Usage Control Adapter are that in the latter some modules are not used (DMO Engine and Bundle Manager) and the DSA Adapter Front End is re-configured in a Service Usage Control Adapter Front End with an embedded Policy Store (PS). The PS keeps the usage control policies for the analytics services.

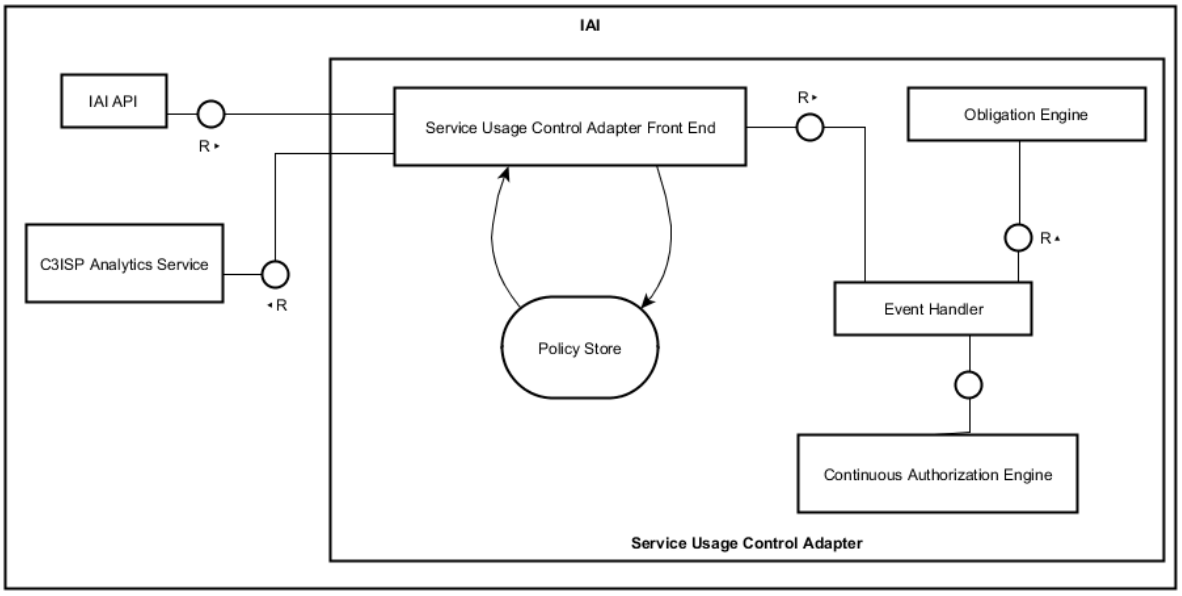


Figure 39: Service Usage Control Adapter

6.2.1. Implementation and Integration Status

At M24 the Service Usage Control Adapter has not been implemented yet and will be part of the next cycle of activities.

The major goals for the final prototype are the development of a system able to control operations that are dynamically performed on data.

6.3. Interface to Legacy Analytics Engines

Legacy analytics engines come in different forms and thus provide different types of interface. One may only be accessed and consumed through its graphical user interface (e.g. visual analytics tool) while the other may provide RESTful or Java API to consume its analytics functions (e.g. Apache Spark). Legacy engines may also put some constraints regarding on how and from where their services can be accessed, e.g. the engine may have been deployed within protected environment which requires adequate security measures and policy to be in place. It is therefore important for the C3ISP framework to offer high flexibility in providing access to such legacy engines in order to maximise its utilisation.

The IAI provides a RESTful API to invoke the legacy analytics services as well as to prepare the data in specific format required for consumption by the legacy engines. The invocation method will depend much on the type of interface supported by each legacy engine. If the legacy engine is a web-based tool with its own graphical user interface the invocation may usually take form of redirection of HTTP requests/responses between service consumer and the legacy engine. In case the legacy engine has its own RESTful or Java API, the IAI API will act as a service wrapper of existing legacy analytics services; this could also mean that an analytics service exposed by the IAI API to C3ISP consumers may make calls to one or more analytics functions supported by the legacy engine's API.

6.3.1. Implementation and Integration Status

The **SATURN Visual Analytics** tool is being integrated into the C3ISP Framework as our first legacy analytics engine; it will provide visualisation of security insights extracted from the shared/merged security data as well as the results of C3ISP-own analytics service. SATURN is a web-based tool with its own graphical user interface to allow interactive usage. The tool is provided by BT, and due to its licensing agreement, the tool can only be deployed within protected BT network environment.

At M24 an OpenVPN[19] server has been set up at the respective BT network domain in order to allow secure tunnelled access from any client's or service consumer's machine via public Internet. As a prerequisite, an OpenVPN client software with the corresponding client's key needs to be installed on the machine requiring access to the SATURN tool. We have established a semi-permanent OpenVPN tunnel between the IAI machine (iaic3isp.iit.cnr.it) and the respective BT network gateway. This VPN tunnel can be used in two ways:

1. To allow IAI API seamlessly redirect any service request to SATURN which originates from any service consumer's machine that doesn't have its own OpenVPN client software and credentials, and
2. To provide the SATURN tool with a secured (read) access to the Virtual Data Lake (VDL) instance which stores the data that have been prepared earlier by IAI API. This access has been successfully tested by manually configuring a new data source in SATURN using the newly created VDL instance's URI (this is the VDL created by the Buffer Manager, see 5.3). It is up to the C3ISP consumer (e.g. a Pilot component) whether or not to automate such configuration task depending on its use cases.

Furthermore, the SATURN tool has been configured to make use of C3ISP Identity Manager (i.e. Open LDAP, see 8.1) for authenticating and authorizing its users according to their roles.

[19] https://openvpn.net

This will ease the identity management within the C3ISP subsystems when examining and enforcing the DSA policies based on the user's roles and attributes, e.g. when preparing and reformatting the data for the VDL.

The major goals for the final prototype at M36 are:

1. To implement the HTTP redirection service at the IAI API which will use the semi-permanent OpenVPN tunnel for providing legacy analytics service access to any authorized C3ISP users. Such users need to be authenticated using the C3ISP Identity Manager;
2. To provide access to a legacy analytics engine that supports either RESTful or Java API for consuming its analytics services. Potential candidate is the Apache Spark engine that is capable for processing and analysing data stored in Big Data platform such as Hadoop.

6.4. Virtual Data Lake

The **Virtual Data Lake (VDL)** is the dedicated service instance for storing CTI data in raw format to be consumed by a Legacy Analytics Engine. We have selected *Apache Hadoop Distributed File System (HDFS)* as a basis to implement this module. In order to ease the integration at initial stage we also selected a *Relational Database Management System (RDBMS)* such as MySQL to implement VDL. MySQL is supported by our first legacy analytics engine, i.e. the SATURN tool.

6.4.1. Implementation and Integration Status

A new VDL can be instantiated by calling the *prepareData* function of the ISI's Buffer Manager API. The Buffer Manager's service will be called by the IAI API as part of its legacy analytics service invocation process. Section 5.3 provides more details on the Buffer Manager API. At M24 each VDL instance is created as a new database within a MySQL database system that is co-located on the IAI machine (iaic3isp.iit.cnr.it). The raw CTI data is populated into a new table of that database and can only be accessed using specific credential (username/password pair). The database connection details and credentials are provided as a URI (i.e. JDBC[20] URL) to be used later for configuring a new data source in the SATURN tool. Following is an example of such VDL-URI:

```
jdbc:mysql://iaic3isp.iit.cnr.it:3306/vdl_u_2688f7f6c2cf4c?useLegacyDatetimeCode=false&serverTimezone=Europe/Rome&usr=u_2688f7f6c2cf4c&psw=0ebc44ed&table=table_bf7d3199_7a97_4f41_bc41_6f1e724e8198
```

In case a VPN tunnel is used to access the VDL instance the corresponding server address will be replaced with a local IP address by which the IAI machine can be contacted from the other end of the tunnel. The Buffer Manager API also supports a function to release the data stored in the VDL, i.e. removing the database and its associated credentials, once the data have been consumed and are not required any longer. This removal process could be mandatory in order to meet the obligation rules specified in the corresponding DSA policy.

The major goal for the final prototype is to implement the VDL in HDFS which can then be accessed by any supporting legacy analytics engines.

[20] Java Database Connectivity

6.5. IAI API

The **IAI API** is the C3ISP front-end for interacting with the analytics services engines (both C3ISP-aware and legacy/already existing analytics services). It exposes methods for invoking analytics services on a DPO data (or multiple DPOs) shared and provided by the Prosumer(s). For this reason, the API includes:

- A method for invoking an analytics service (*runAnalytics*), which will receive as input the service analytics name (in the form of a unique identifier) and the DPO identifier (DPO-Id) to be involved in the analytics. The analytics name is an identifier specifying an analytics service available in the analytics engine;
- A method for processing multiple DPOs, i.e. two or more DPOs can be involved as input to an analytics service. The method is analogous to the previous one, but it receives as input a list of DPO-Id.

In the latter scenario, we assume that the DSA associated to each CTI is the same: this is the reasonable scenario where several Prosumers share CTI data, by using their agreed DSA, and that want to perform analytics on them.

Depending on the kind of analytics service, the IAI API will be in charge of triggering the creation of the VDL (for the Legacy Analytics Service), by performing the required operations against the ISI (e.g. Read DPO) to feed the VDL instance. We also foresee the possibility of having a specific API (or a specific analytics service) that will create the VDL instance without specifically running an analytics job (i.e. a *void analytics service*): this will be used by a Legacy Analytics Engine that will need to interact directly with the sanitised data, e.g. for visualisation purposes where it has to navigate or drill down into the stored information in a dynamic way.

6.5.1. Implementation and Integration Status

At M24 the IAI API offers a first set of analytics aimed at addressing analytics requests from the four pilots. The available API are used to invoke analytics related to WP2, WP3, WP4 and WP5 activities, moreover, a generic **RunAnalytics** function has been added as a generic interface to call analytics. The APIs accept as input a JSON string reporting all the parameters that are specific for the invoked analytics. The RunAnalytics accepts a JSON where the first parameter is the "Analytics Name", which allows the indirect invocation of the available analytics. More analytics are being added and linked through the IAI API as they are developed:

- **detectDGA**: it takes domain-name logs and checks if they are DGA (Domain Generated Algorithm) using a third-party algorithm;
- **matchDGA**: it takes domain-name logs and checks if they are DGA using a public source of known DGAs;
- **runAnalytics**: generic method to invoke any analytics available in the IAI API, which takes as input the name of another analytics to invoke and the set of parameters;
- **spamEmailClassify**: analytics that takes as input a set of spam emails in *eml* format and returns the class assigned to each email, based on the spammer goal (e.g. phishing, advertisement, etc.);
- **spamEmailClusterer**: analytics that takes as input a set of spam emails and divides them in clusters based on structural similarity;
- **spamEmailDetect**: takes as input one or more *eml* files and separates them in genuine emails (Ham) and unsolicited ones (Spam) sets;

- **analyseMaliciousHost**: analytics that takes as input a set of security logs such as IDS/IPS alerts, Malware alerts or Web Proxy logs in *csv* format and returns a prioritised list of malicious hosts based on specified criteria (e.g. frequencies of occurrence, commonness, threat level, etc.);
- **analyseDNSTraffic**: analytics that takes as input a set of DNS request logs in *csv* format and returns a list of domain names with suspicious network traffic behaviours (e.g. high frequency of requests);
- **findVulnerability**: analytics that searches in an archive of CVE, stored in the DPOS the keyword passed as a parameter;
- **findMalware**: analytics that searches, in an archive stored in the DPOS, information about the malware related to the keyword.

Complete description of the available APIs is reported in deliverable D8.2.

The major goals for the final prototype are the inclusion of all analytics needed to address the pilots' requirements. Furthermore, a set of generic analytics which are not pilot specific will allow generic users of the IAI API to run simple data analysis operation on provided or stored data.

6.5.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the IAI API:

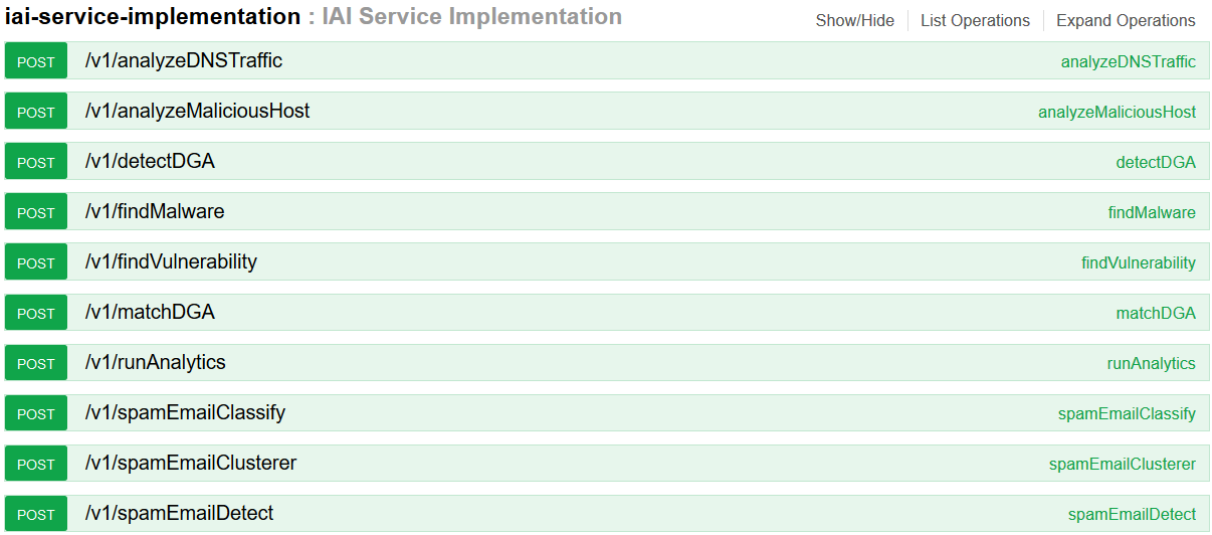


Figure 40: IAI API from Swagger UI

7. Subsystem Updates & Status: DSA Manager

The DSA Manager is an autonomous subsystem of the C3ISP Framework appointed to provide services for the definition of policies in the DSAs, creation, storage and management of DSAs to the Prosumers.

This section reports on the revised DSA Manager that has been developed at M24 and deployed into the C3ISP Test Bed.

The DSA Manager is made up of the following components, described in details in the next sections:

- DSA Editor: to write the sharing rules that can be understood by humans;
- DSA Mapper: to translate the sharing rules to an enforceable language that can be processed by a machine;
- DSA Store API: to manage the DSAs from the other components;
- DSA Manager Gateway: to provide DSA services to other C3ISP subsystems;
- DSA Store: to persist the DSAs on a storage area.

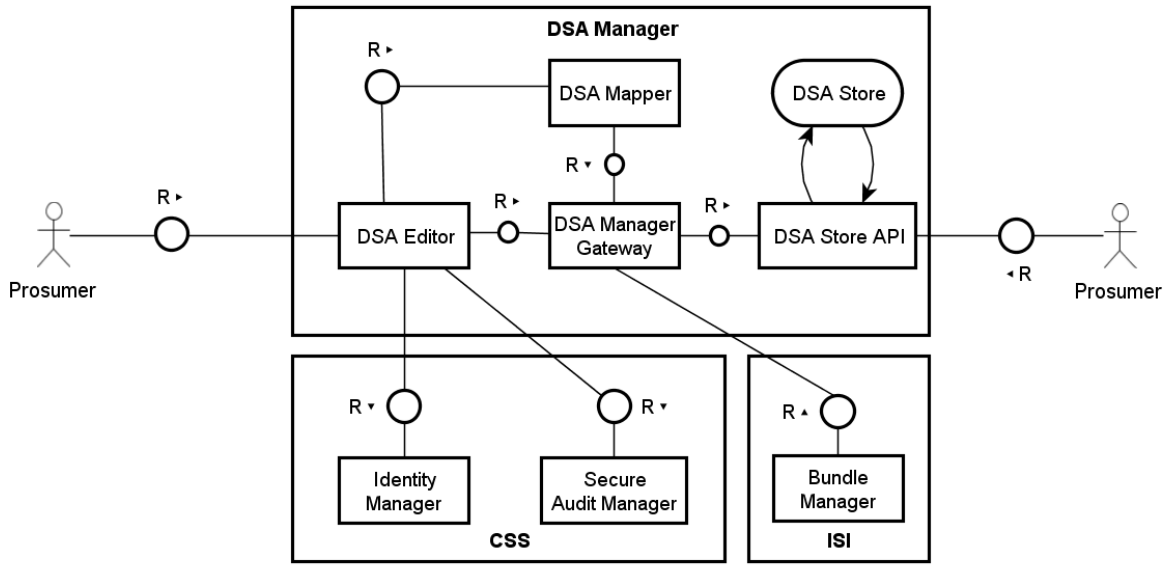


Figure 41: DSA Manager

The DSA Manager interacts with external clients:

- the Prosumer, which would use the services exposed by the DSA Editor and (optionally, if needed) DSA search capabilities exposed by the DSA Store API;
- the Information Sharing Infrastructure (ISI), that needs to use the DSA Manager Gateway for supporting the data sharing among the Prosumers using the C3ISP Framework;
- the Common Security Services (CSS) for secure auditing of its activities and for identity management.

7.1. DSA Editor

This component has been designed in detail during implementation and the following diagram describes its internal modules and the interaction with the other DSA Manager's components:

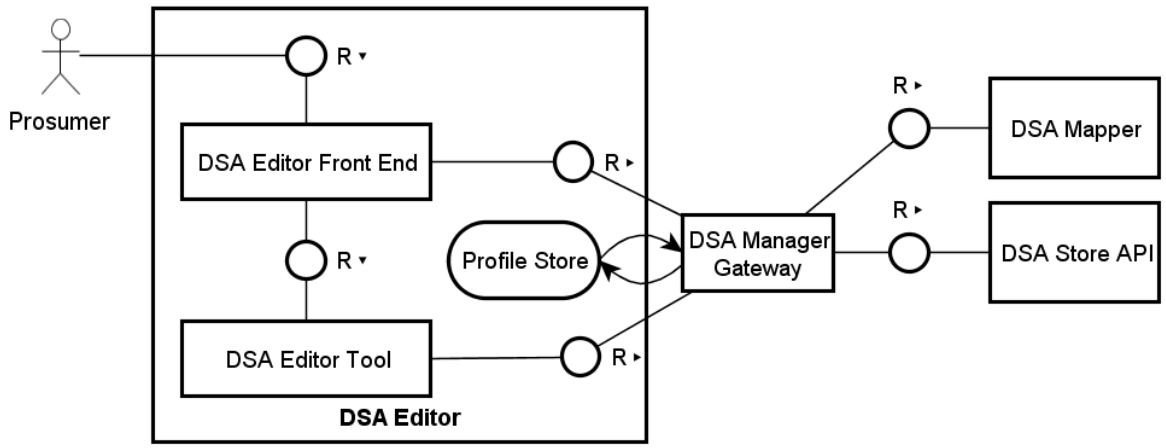


Figure 42: DSA Editor

The component is made of the following modules:

- DSA Editor Front End: this is the entry point that allows Prosumers to access the available DSAs via a role-based access control system, where each user can work on his/her own DSAs without impacting other users. It interfaces with the DSA Editor Tool to allow the creation of DSAs and with the DSA Manager Gateway to manage the users' Profile Store, including some DSAs functions (in particular the "revoke DSA" and "delete DSA" operations);
- DSA Editor Tool: a user-friendly and easy to use web application that allows creating both DSA Templates (a generic DSA with predefined useful policies for a certain context that must be instanced to be used) and of DSA instances (a DSA specifically created from a DSA Template by adding rules for refinement that can be directly used for enforcement): see D8.1 for more on DSA and DSA Templates;
- Profile Store: this is a repository that hosts the users that can access the other modules with their roles and a reference to the DSAs they have written.

A very detailed description about the usage and appearance of the DSA Editor is contained in D8.3.

7.1.1. Implementation and Integration Status

At M24 the DSA Editor has been enhanced to support new vocabularies based on a single common ontology defined for the Pilots. Further, it has been extended to allow specifying terms to represent DMOs with parameters and options (e.g. as required by the Anonymization Toolbox, e.g. *AnonymizeBySuppression*) and Data Analytics services (e.g. *InvokeDetectDGA*). Current supported DMOs and Data Analytics services, as well as how they have been implemented, are reported in D8.3.

The DSA Editor now is fully integrated with the other components of the DSA Manager subsystem.

As anticipated, the DSA Editor keeps user information on a Profile Store. This is implemented as a relational database on MySQL. Its schema is straightforward and is shown next:

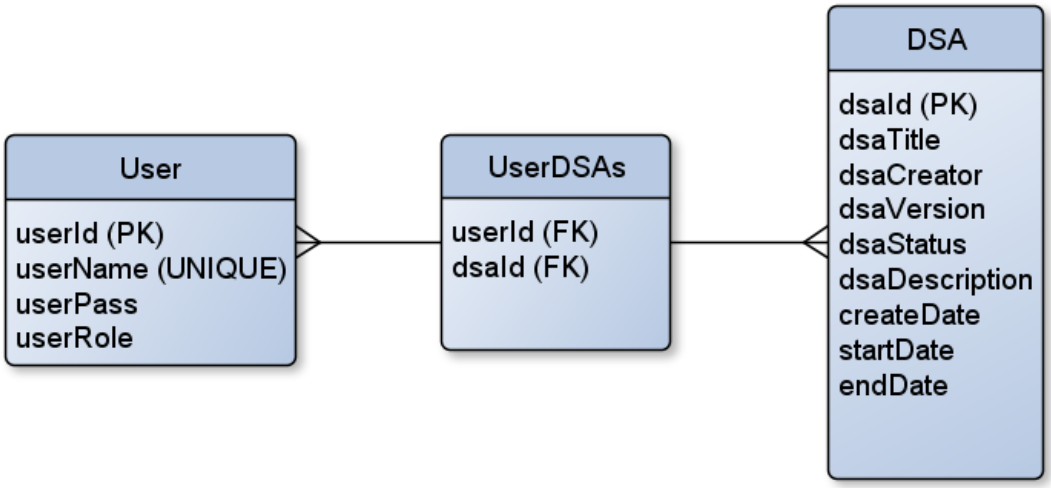


Figure 43: User Profile relational database schema. There is a 1:n relationship between a User and its DSAs, and potentially, a DSA can be seen by many users, which is why we added a table for the user-DSA relationship.

Please refer to D8.3 for an in-depth description regarding the maturation of this component. For the next period, we will focus on enhancing the usability of the DSA Editor Tool module, making more user friendly for the end-user, as we will also get feedbacks based on the Pilots evaluation.

7.2. DSA Mapper

The **DSA Mapper** is responsible for translating the DSA policies from the Controlled Natural Language (CNL) employed by the DSA Editor into a low level directly enforceable policy language (XACML-based called UPOL). It is called by the DSA Editor to translate the DSA policy before the DSA will be persisted to the DSA Store.

The current status of the DSA Mapper functionalities maturation and how they work are described in D8.2.

The improvements of the DSA Mapper within the second year of the project are mainly related to the new functionalities implemented by the DSA Editor to manage new kind of policies or different structure of policies or vocabulary.

Hence, the architecture of the DSA Mapper component is still the same but its translating capabilities are enhanced in terms of accepted vocabularies, and kind of policies that can be mapped from CNL to UPOL/XACML.

7.2.1. Implementation and Integration Status

At M24 the DSA Mapper is fully integrated in the architecture. It can be called by the DSA Editor and it is able to map the DSA written by using the DSA Editor. It persists the mapped DSA by using the DSA Store API.

The DSA Mapper is implemented via a RESTful web service that provides the following API:

- **MapDSAByID**, which takes as input a DSA identifier, fetches the corresponding DSA and maps all the policies creating the mapped DSA (a DSA that contains both the CNL statements and the UPOL representation in a single XML container);
- **buildUPOL**, which tasks as input a DSA identifier and returns the enforceable UPOL representation. This DSA representation is then used for pairing it with the

CTI data. This API is part of the Bundle Manager workflow, which calls it via the DSA Manager Gateway APIs.

As described more in detail in D8.2 Section 4.2, the DSA Mapper has been improved to be able to manage the new capabilities of the DSA Editor. In particular, to:

- *Support new vocabularies for the C3ISP pilots' context*: it is able to learn new vocabularies to manage policies related to the four C3ISP's pilots;
- *Support definition of Data Manipulation Operations* (DMOs) *(pre/post-processing rules)*: the DSA Mapper is able to recognize DMO policies and to translate them as obligations policies in which the subject is always the term "System";
- *Support for Data Analytics Operations*: the DSA Mapper translates analytics functions as "Actions";
- *Translate policies on Analytics Results*: policies related to data derived from analytics functions are in a separate section of the DSA and the DSA Mapper translates them in a separate section too. Note that the current implementation of the DSA Mapper manages policies on first order derived data, i.e., on data generated as results of the application of analytics on original data, the one the DSA is related to.

In the near future, the DSA Mapper will be extended to also manage policies on data derived from derived data (second order or deeper derived data).

7.2.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the DSA Mapper:

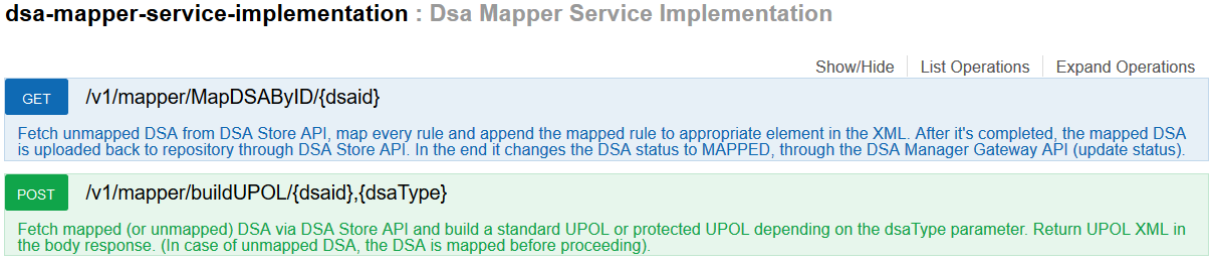


Figure 44: DSA Mapper API from Swagger UI

7.3. DSA Store API

The **DSA Store API** is the external interface of the DSA Store subsystem, and it provides functions for managing the storage of DSAs. The DSA Store is used by the ISI to retrieve DSAs or by the DSA Editor when it needs to create or update DSAs.

The DSA Store API supports the following interface:

- **Create DSA**: used to persist a DSA in the DSA Store. The API returns a unique DSA identifier;
- **Read DSA**: used to retrieve a DSA by its DSA identifier;
- **Update DSA**: used to modify the content of an already existing DSA, by its identifier;
- **Delete DSA**: used to delete a DSA, by its identifier;

- **Search DSA**: given a JSON-based search string (more in D5.3), query the DSA metadata repository, and return a set of metadata entries corresponding to the matching DSAs (alternatively, a set of DSA IDs, depending on an input parameter). DSA metadata is stored in a JSON document, and contains a set of DSA fields extracted from the DSA XML files.

The DSA Store API has been redesigned to support Create, Delete, and Read of DSAs, as well as the new Search functionality. The search functionality was added to support managing multiple DSAs, and allowing Pilot applications to retrieve only those DSAs which are relevant to the Pilot user and application.

Functionality related to manipulating DSA status, validity and enforceable policies is now available through the DSA Manager Gateway. The following functionality has been removed from DSA Store API to another component: *Retrieve the status of a DSA*, *Check DSA Validity*, *Fetch enforceable policies from the DSA*, *Revoke a DSA*, *Delete enforceable policies from a DSA* and *Add/Update enforceable policies in a DSA*.

7.3.1. Implementation and Integration Status

At M24, the prototype of DSA Store API is fully functional and integrated with the DSA Store (See Section 7.5). Its REST API is deployed at the C3ISP test environment[21].

For storing a DSA into the DSA store, a pair of two corresponding JSON documents are created in the DSA Store: one is the DSA itself and another is the DSA metadata, which are linked by the DSA-Id. The DSA metadata contains a set of DSA fields extracted from the DSA XML files and it is used for the sake of Search DSA.

In fact, in addition to the Create, Read, Delete functionalities described during the design phase, the prototype also supports searching on a set of DSA fields using JSON-based query format. The format of these queries is identical to that defined for the Data Protected Object (see D8.2), with a DSA-specific set of terms.

7.3.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the DSA Store API:

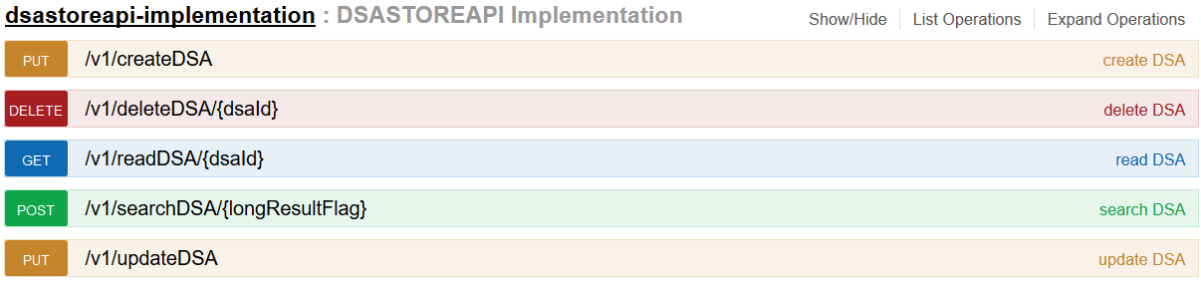


Figure 45: DSA Store API from Swagger UI

7.4. DSA Manager Gateway

The **DSA Manager Gateway** was formerly an internal module part of the DSA Editor. It has been extracted and promoted to a standalone component because its services are useful not only to the editor itself, but also to the other DSA Manager components and to the ISI subsystem. In fact, the DSA Manager Gateway is a list of APIs that the other components can use to interact with the DSA Mapper and the DSA Store at more logical level. In particular, it provides an

[21] https://dsamgrc3isp.iit.cnr.it/dsa-store-api/

interface that contains more business than the DSA Store API, which is a direct CRUD interface to the DSA Store (e.g. the DSA Manager Gateway can return parts of the DSA, instead of the whole DSA returned by the DSA Store API).

The DSA Manager Gateway supports the following interface:

- **Retrieve the status of a DSA** (see DSA status in section 3.1 of D8.1), by interacting with the DSA Store API;
- **Add/Update enforceable policies in a DSA**, given the DSA identifier: used to add and update enforceable policies (i.e. the policies created at the DSA Mapper level, see section 7.2) contained in a DSA;
- **Fetch enforceable policies from the DSA**, given the DSA identifier: used to extract enforceable policies from a DSA;
- **Delete enforceable policies from a DSA**, given the DSA identifier: used to delete enforceable policies from a DSA;
- **Check DSA Validity**: used to verify if the DSA is in a valid state (e.g. not expired, not revoked, etc. See D8.1);
- **Revoke a DSA**: as explained in the D8.1 section 3.1, the DSA can be revoked for certain reasons (e.g. the parties involved in the agreement decide that it is no longer valid), so it should be possible to revoke a DSA given its identifier.

7.4.1. Implementation and Integration Status

At M24, the DSA Manager Gateway is fully implemented. Its REST API services are deployed at the C3ISP test environment. The component is stable and we do not foresee any major change in the future. However, since we moved DSA Manager Gateway out of the DSA Editor, it shows many internal APIs that we might need to refactor in order to reserve them for DSA Editor use only.

7.4.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the DSA Manager Gateway:

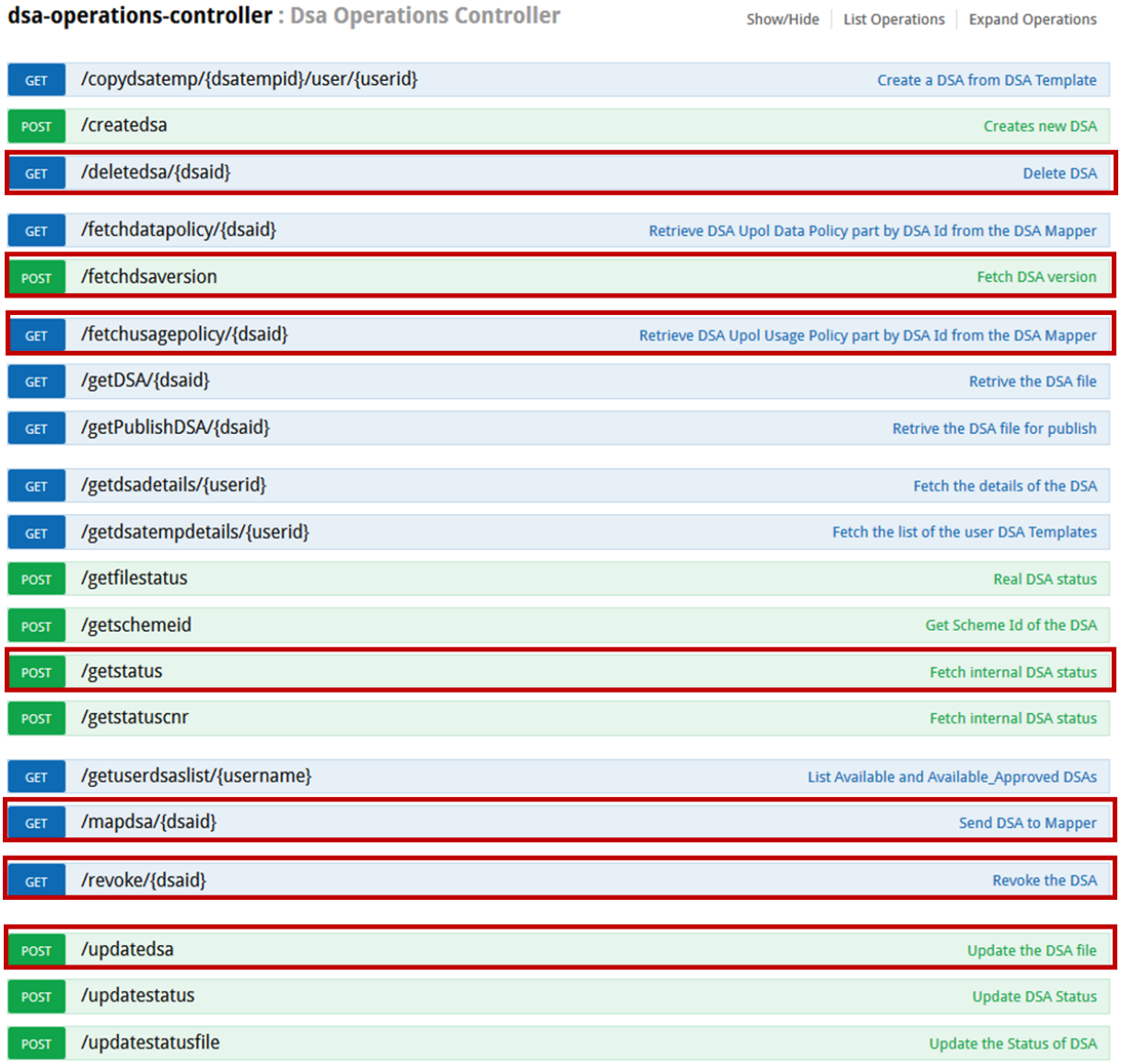


Figure 46: DSA Manager Gateway from Swagger UI

As we said in the previous section, there are also some internal-use only APIs. The ones highlighted in red are those used for implementing the API listed in 7.4.

7.5. DSA Store

The **DSA Store** is a database that acts as a repository where the DSAs are stored and consumed by the DSA Store API.

7.5.1. Implementation and Integration Status

At M24 the DSA Store is fully implemented and working. It has been implemented with an instance of MongoDB, a NoSQL database. MongoDB uses the concept of collections, which is a group of document, much like the same a relational database have tables and records (tuples). Collections are stored into MongoDB databases.

We defined a dedicated MongoDB database called “dsastore” with a collection called “dsaCollection”. The documents in the “dsaCollection” are the DSA and are stored by the DSA Store API. As any MongoDB collection, the “dsaCollection” is schema-free, which means that its schema is dynamic and is created per each document (i.e. DSA) when it is stored.

In particular, as described in 7.3.1, for each stored DSA a pair of documents are created in the “dsaCollection”: one for the DSA itself, and one for the DSA metadata (used to support the search function of the DSA Store API). The next screenshot shows an example of these two documents, as they are stored in MongoDB:

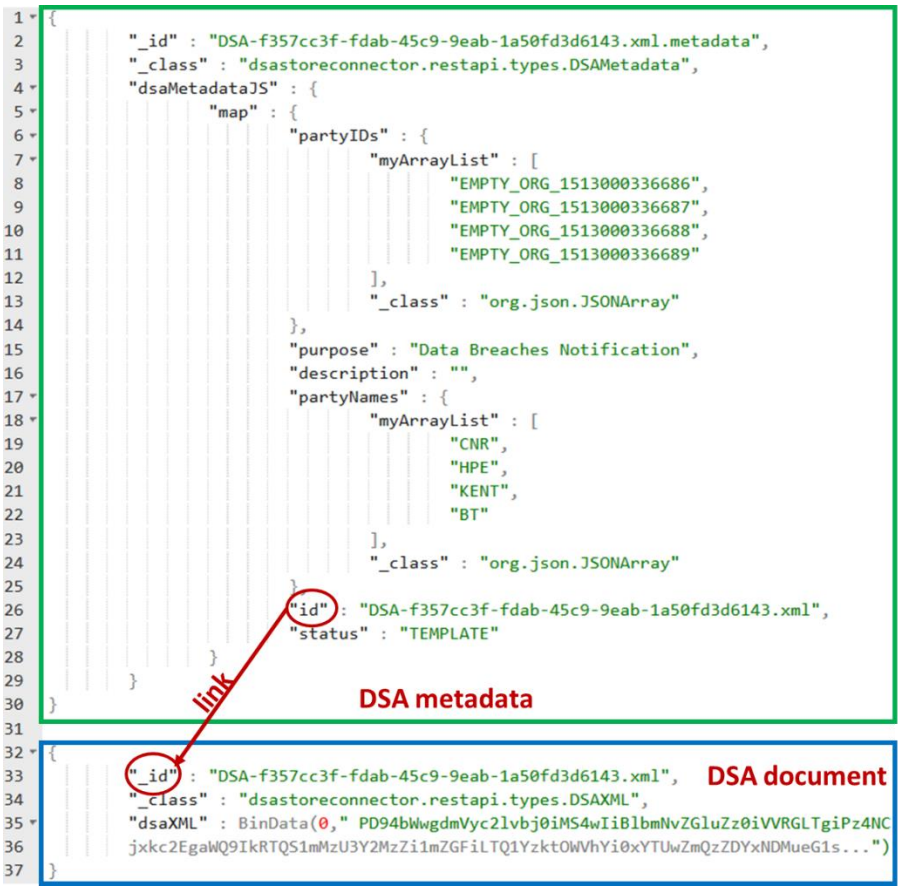


Figure 47: the DSA metadata part (JSON on the top) and the DSA document (JSON on the bottom, partial). The two MongoDB documents are linked by the “id” and “_id” fields

MongoDB is set up for requiring user authentication and a dedicated service account has been defined to allow the DSA Store API to connect to the “dsastore” database.

8. Subsystem Updates: CSS – Common Security Services

The CSS subsystem is made up of the following components, those updated at M24 are described in detail in the next sections:

- Identity Manager: to provide identification, authentication services, users attributes for policy evaluation;
- Key and Encryption Manager: to manage cryptographic keys and encryption services, including functionalities for homomorphic encryption;
- Secure Audit Manager: to allow secure storage of events occurring during the C3ISP Framework operational activities.

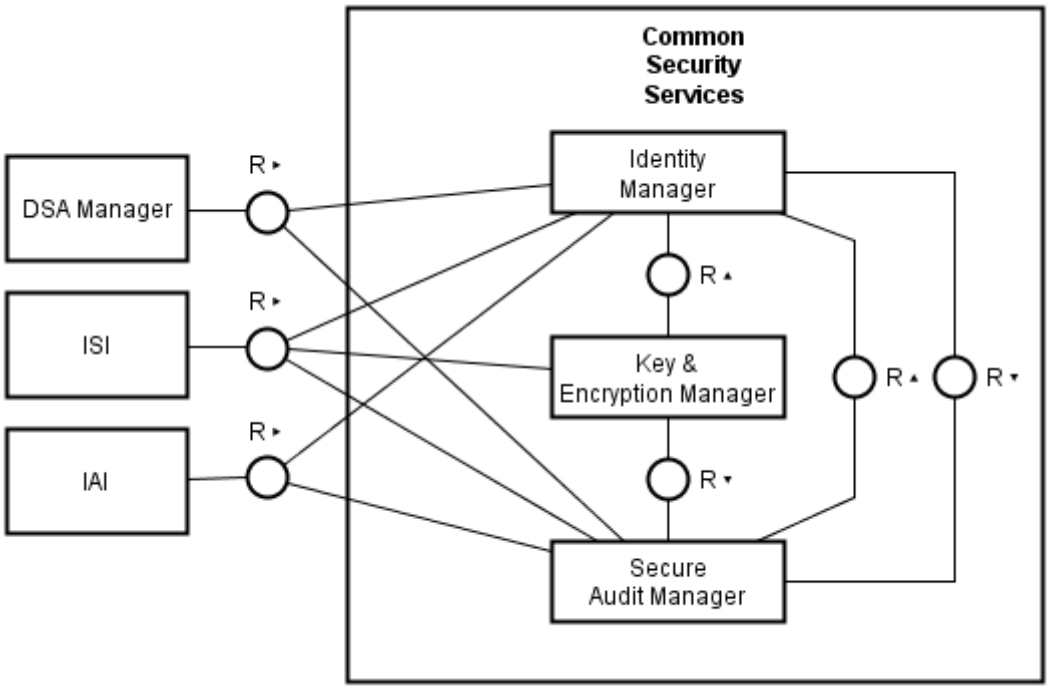


Figure 48: Common Security Services

This section reports on the revised CSS that has been developed at M24 and deployed into the C3ISP Test Bed.

The CSS interacts with external clients:

- The DSA Manager, for identification and auditing purposes;
- The Information Sharing Infrastructure for identity-related purposes, key and encryption services and auditing;
- The Information Analytics Infrastructure for identity-related purposes and auditing.

8.1. Identity Manager

The **Identity Manager** aims at identifying entities and storing authorization information within C3ISP.

8.1.1. Implementation and Integration Status

At M24 the Identity Manager based on an LDAP service is fully deployed, running and working properly. It holds user entities that can be used for authentication. The user attributes can be used in DSA policies, since the Identity Manager is integrated with the Attribute Managers described in 5.1.3.1. We also setup group-based memberships, which allows us to defined DSA policies that uses groups (as reported in D8.2 in the DSA Editor section).

We have setup also group membership for the Enterprise and SME pilots, which will use DSA policies based on this authorisation check, as shown in the next figure:

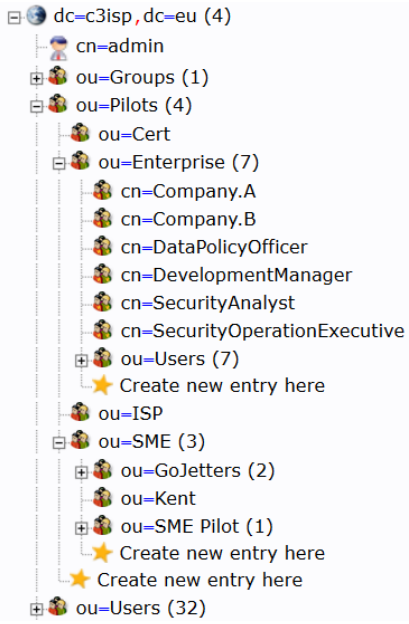


Figure 49: Pilots' LDAP structure

In the next period, we will continue to support Pilots integration activities by refining the Directory Information Tree structure and user attributes, should they have the need for their own authorisation DSA policies. In particular, Pilots like ISP and CERT may support standard for authorization such as, **Oauth2** [23] and/or **SPID** [24] to ease the interaction from stakeholders with C3ISP Framework.

8.2. Key and Encryption Manager

The Key and Encryption Manager (K&E Manager) is responsible for key management and encryption services needed for ensuring the confidentiality data throughout the C3ISP Framework. Our goal at M24 during component implementation is to save time by using a suitable pre-existing open source software and to enable to integrate innovating features, mainly homomorphic encryption mechanism, in such software. By that way, the following requirements from D7.2 were clarified and they should be taken into account during the choice of a free open-source software:

- Support key generation with classic cryptosystems like AES, RSA or Elliptic Curve Digital Signature Algorithm (ECDSA);
- Support key management, i.e. the tool can guarantee authenticity, integrity and confidentiality of secret and public keys management;
- Support key deletion;
- Support key versioning;
- Provide regular updates and maintenance.

Among the existing solutions we have evaluated, we put in short list the ones offering regular updates, which are the following: Key Management Interoperability Protocol (KMIP), OpenStack™ Barbican, and HashiCorp Vault.

- KMIP: some features in KMIP are released as open protocol, but the most of them are not. The last KMIP version was released in 2015, since then, no implementation or any update has been done. However, it is able to handle AES and RSA, but not the elliptic curve-based mechanism;
- OpenStack™ Barbican: this is the key management component for OpenStack (one of the main open source solutions for cloud computing). Barbican meets all the requirements and recommendations. Thanks to it being integrated in the OpenStack project, it is updated every 6 months. The desired cryptosystems AES, RSA, and ECDSA are implemented. However, the main disadvantage of Barbican is its dependency on OpenStack, it means that an entirely dedicated machine should be reserved to run Barbican. In particular, this makes it difficult to have a fully local installation;
- Vault: this is part of a software suite developed by HashiCorp for distributed architectures. Vault is an interesting solution because its installation is simple (only an executable file to download and run in standalone way). Moreover, many API in various programming languages exist. Especially, the Vault features implemented in Java are regularly updated from the community. However, poor documentations and few tutorials for Vault usage are the main disadvantages of this technology.

After considering advantages and disadvantages in each of these three technologies, we finally decided to choose Vault for implementation of K&E Manager. According to Section 7.2 of Deliverable 7.2, Vault is used for supporting two modules namely Data Protected Object (DPO) – Key & Encryption Manager and Full Homomorphic Encryption (FHE) – Key & Encryption Manager.

8.2.1. Implementation and Integration Status

With respect to the C3ISP high-level architecture defined in Section 7.2 of Deliverable 7.2, the Key & Encryption Manager interacts with the C3ISP subsystems and components as in the following figure:

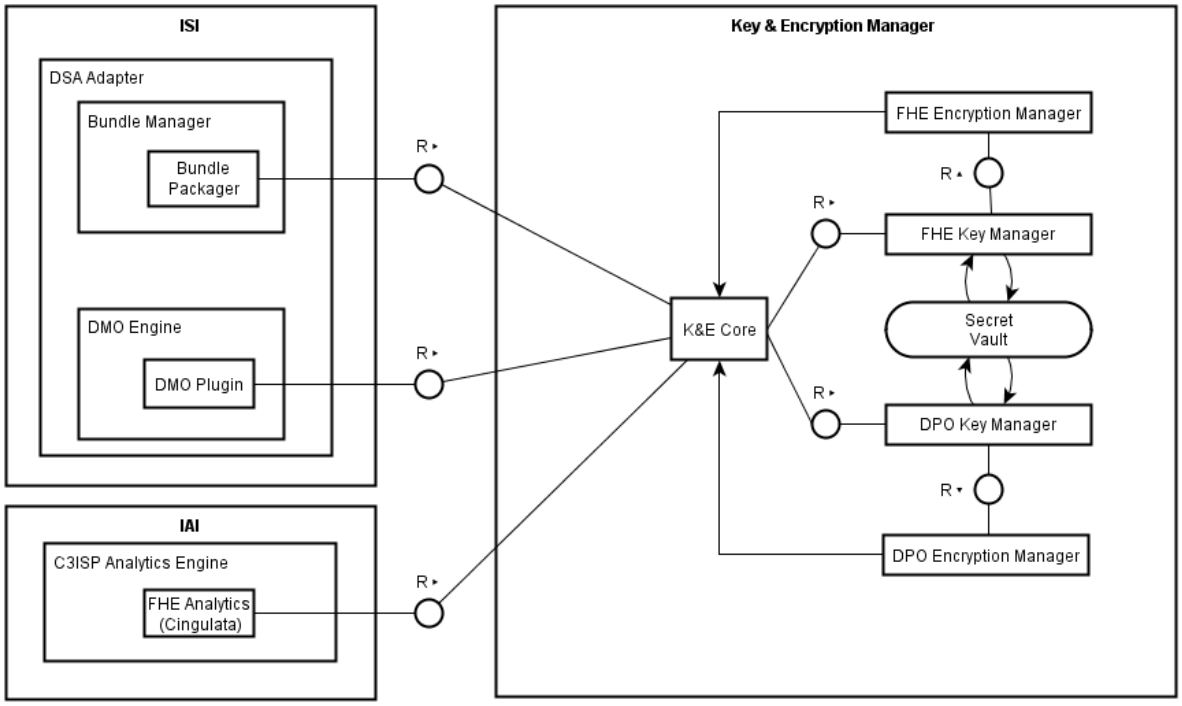


Figure 50: Key & Encryption Manager architecture

8.2.2. K&E Core

At M24, K&E Core APIs was almost done. These APIs allow dispatching the requests from the ISI (specifically the Bundle Packager or the DMO Plugin for transcrypting preparation) and from the IAI (specifically the FHE Analytics).

Table 34 – Software requirements

Specification K&E Core API	Status	Description
Integration with ISI – Bundle Packager	100%	Using AES encryption, it allows to encrypt and decrypt the packet of data protected object (DPO) during the process of creating, updating and deleting. The key management platform for **One Pilot One Key** was done, which is mainly identified by DSA ID.
Integration with ISI – DMO Plugin for transcrypting preparation	80%	The major tasks are: • Developments for key management, by using Kreyvium encryption, which is a lightweight homomorphic-friendly cryptosystem; • Selection of the correct keys and parameters for transcrypting Kreyvium – encrypted data to FHE encrypted data.

To satisfy the C3ISP requirements, we designed RESTful APIs which are supporting the following functionalities:

1. The Advanced Encryption Standard **(AES)**: a widely used specification established by the American government. We do not use RSA, because of its performance and its practices;
2. A Fully Homomorphic Encryption (**FHE**) scheme: we will use an implementation of the Fan-Vercauteren (**FV**) scheme extracted from Cingulata (backend of FHE Analytics, see Section 6.1.2);
3. **Kreyvium**: a lightweight homomorphic-friendly cryptosystem. It is used for transciphering, i.e. turning data encrypted with Kreyvium into the corresponding homomorphically-encrypted data without ever decrypting the data.

The main innovating feature of the K&E Manager we are working on is integrating homomorphic encryption mechanism in the Vault software. Moreover, the K&E Manager can offer a tranciphering feature for integrating with FHE technology, i.e. it allows trans-crypting data from classically encrypted form to homomorphic encryption form for analysing this data with FHE later.

8.2.2.1. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the K&E Core:

Figure 51: K&E Core API from Swagger UI

8.2.3. Key Management: Key Generation & Access

The Key ID is identified by the DSA ID and the desired version number of Vault.

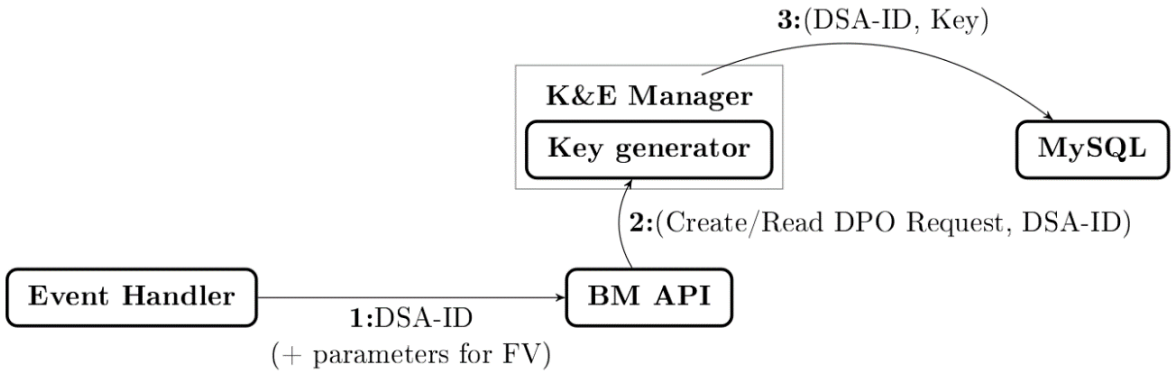


Figure 52: Key generation in general case

8.2.3.1. DPO Key Management

This is the description for the key management of the **DPO Key Manager** module:

Table 35 – DPO Key Manager

Specification of DPO Key Manager API	Status	Description
Integration with K&E Core for key generation and management during data Bundle creation	100%	Using AES encryption scheme, DPO Key Manager API allows generating the public/secret keys assigned to DSA ID (see Section 8.2.3.2)

Key Generation with AES

Vault generates keys automatically using the random number generator of the operating system. A user might call this function several times. Each time they do so, a newly created key set replaces the former one. Vault keeps track of the latest version of the keys and also keeps former versions in storage. Keys are stored on a Vault dedicated MySQL database backend.

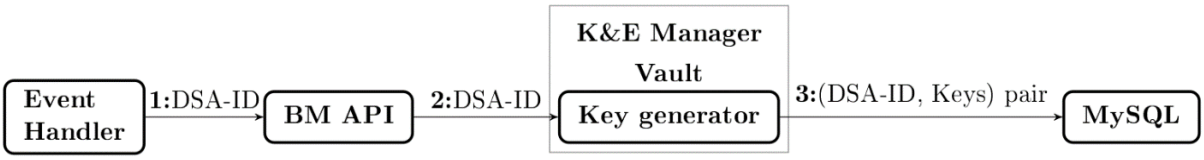


Figure 53: Key generation for using AES.

Key Access

Vault retrieves the latest version of the key set from the MySQL backend on its own. In case the user asks for a key set that does not exist, i.e. this user never asked for any keys to be created before, Vault automatically creates a suitable set.

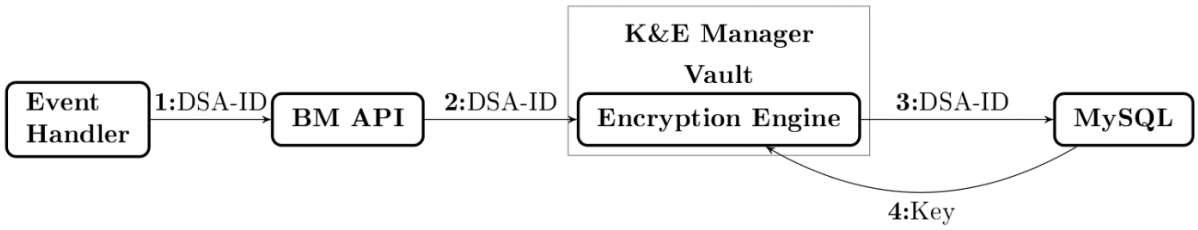


Figure 54: Key access for using AES

8.2.3.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the DPO Key Manager:

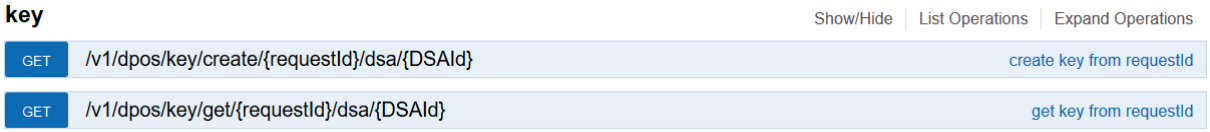


Figure 55: DPO Key Manager API from Swagger UI

8.2.3.3. FHE Key Management

This is the description for the key management of the **FHE Key Manager** module:

Table 36 – FHE Key Manager

Specification for FHE Key Manager API via K&E Core API	Status	Description
Kreyvium key management for tranciphering	100%	Using Kreyvium encryption scheme, FHE Key Manager API allows generating the secret key assigned to DSA ID (see Section 8.2.3.4).
FHE key management	100%	From FHE technology based on FV encryption scheme, this API provides a triple keys (public key, secret key and evaluation key). From each FHE analysis model, FHE Key Manager API provides a triple keys because of security parameters. By that way, each Prosumer which is identified by DSA ID can be in possession of multiple triple FHE keys.

FHE Key Generation

In order to generate FHE keys, we use Cingulata toolchain. In the case of FHE technology, we do not wish Vault keeping track of key versioning. Hence, the function for FHE key generation was implemented in the Java API and we used Vault only for interacting with its MySQL backend.

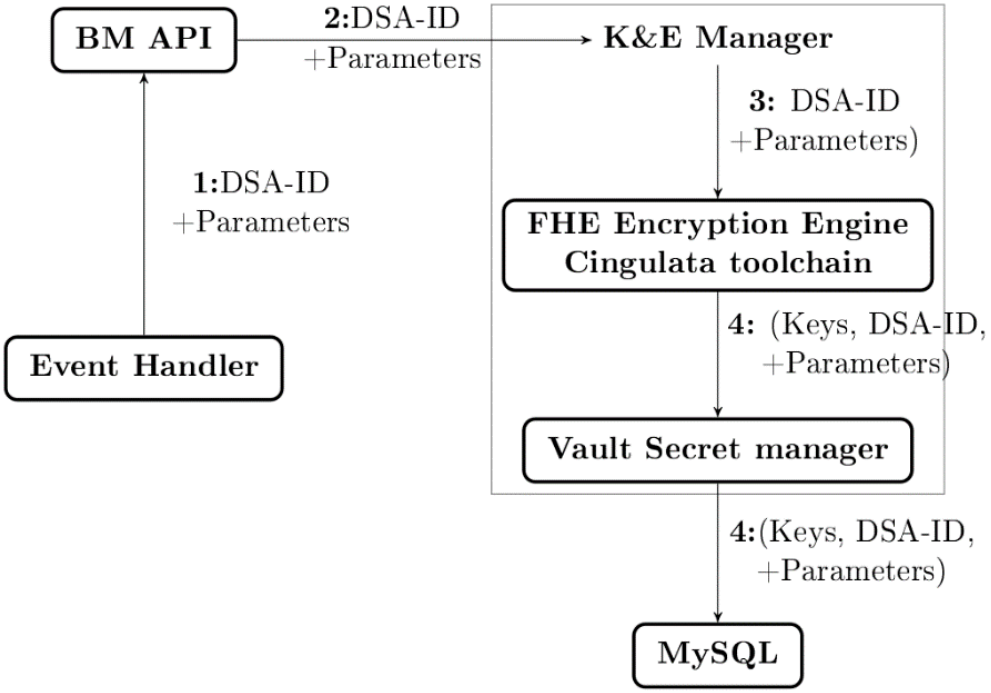


Figure 56: FHE key generation process

FHE Key Access

This is the process for accessing the generated FHE keys which are only applied for decrypting FHE encrypted data, but the secret key is never out of K&E Manager:

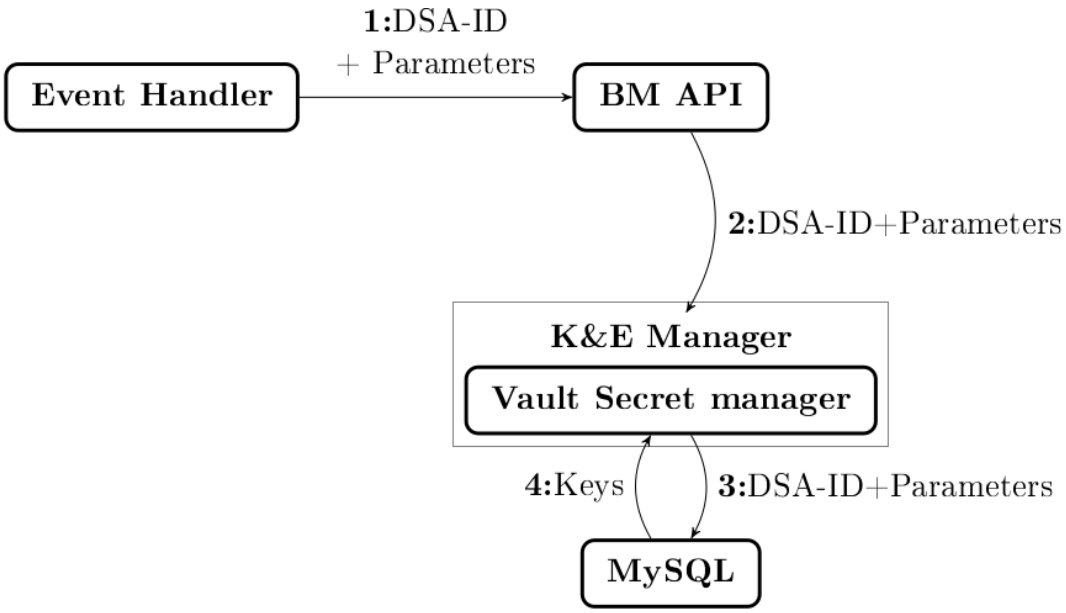


Figure 57: Retrieving FHE key process, only in the case of decrypting FHE ciphertexts

8.2.3.4. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the FHE Key Manager:

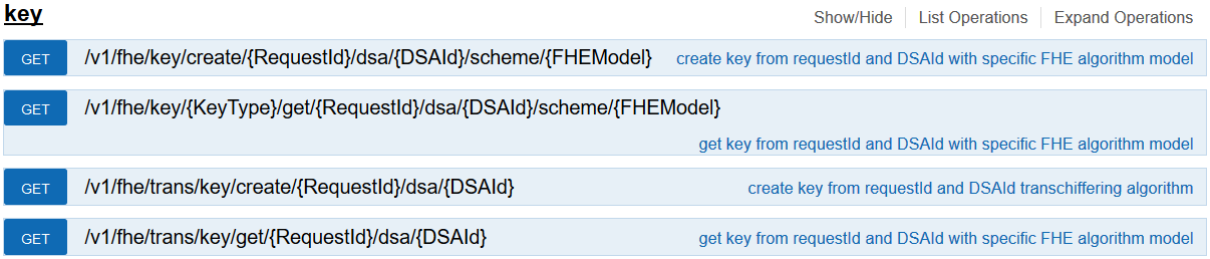


Figure 58: FHE Key Manager API from Swagger UI

8.2.4. Encryption and decryption

The following diagram shows the encryption and decryption process:

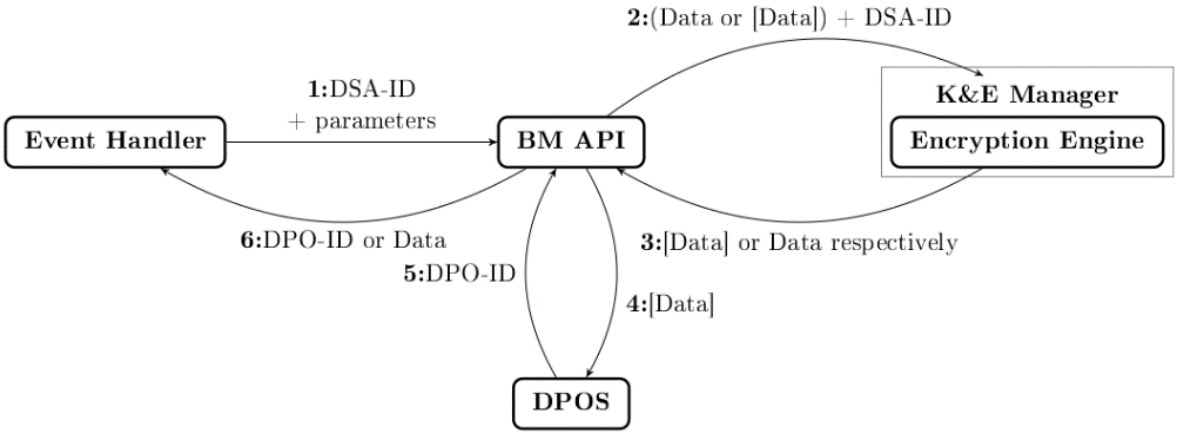


Figure 59: Encryption and decryption processes in general case

Note that Data denotes the clear data, whereas [Data] denotes the corresponding encrypted data. The security parameters for FV cryptosystem are used to determine the circuit multiplicative depth, this parameter is needed to instantiate correctly and securely FV cryptosystem. These parameters are also used for the FHE Analytics and a choice for processing with transciphering if necessary.

8.2.4.1. DPO Encryption and decryption using AES

This is the description for the DPO encryption and decryption functionalities using AES of the **DPO Encryption Manager** module:

Table 37 – DPO Encryption Manager

Specification for DPO Encryption Manager about Encryption/Decryption API	Status	Description
Integration with K&E Core for encrypting or decrypting bundle data	100%	Using AES encryption scheme, the DPO Encryption Manager API allows encrypting or decrypting the bundle data by using DSA ID as key identifier (see Section 8.2.4.2).

The following diagram describes the process:

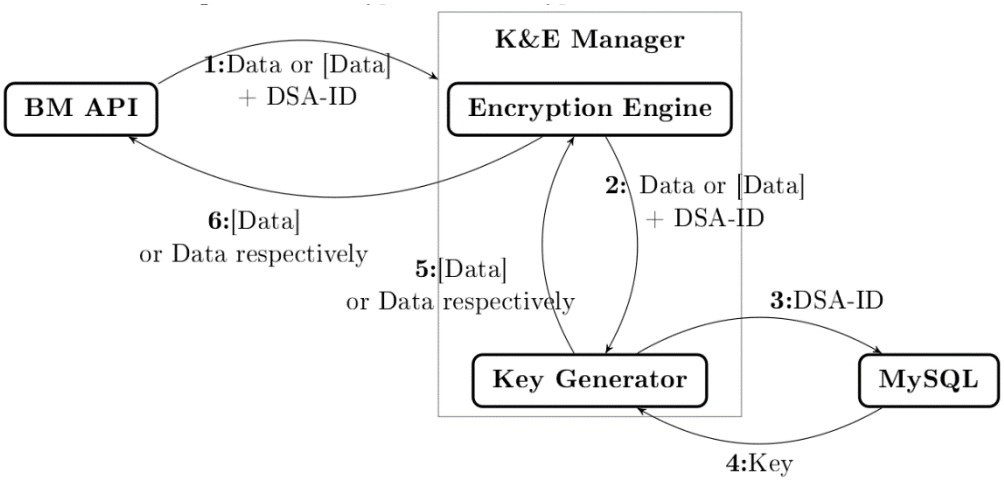


Figure 60: Encryption and decryption for using AES

8.2.4.2. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the DPO Encryption Manager:

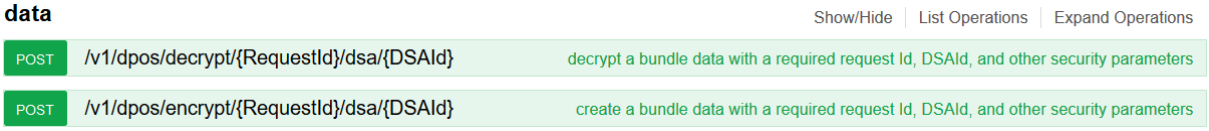


Figure 61: DPO Encryption Manager API from Swagger UI

8.2.4.3. FHE Encryption and Decryption (based on FV scheme)

This is the description for the FHE encryption and decryption functionalities based on FV scheme of the **FHE Encryption Manager** module:

Table 38 – FHE Encryption Manager

Specification for FHE Encryption Manager about Encryption/Decryption API	Status	Description
Encrypting or decrypting data in FHE form	100%	The FHE encryption / decryption API allows decrypting the bundle data by using DSA ID and FHE model as key identifier (see Section 8.2.4.4).
The API for transciphering data to FHE form	80%	The API allows encrypting data with Kreyvium encryption and automatically providing and storing such data into FHE form for analysis with FHE later. At the moment of writing this deliverable, the feature for automatically generating & storing Kreyvium encrypted data to FHE form is not finished yet.

The following diagram describes the process:

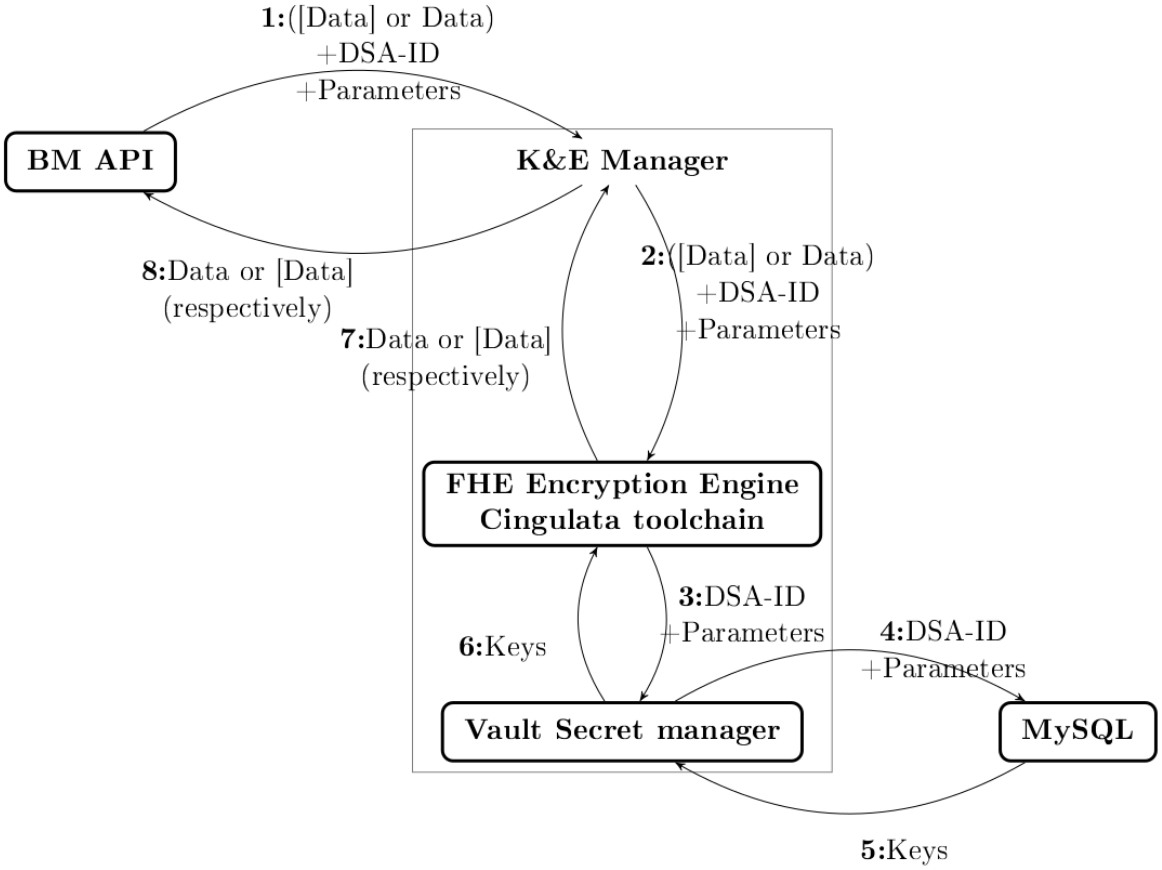


Figure 62: FHE Encryption and Decryption process

8.2.4.4. Published APIs

The following screenshot shows the implemented API that are published for usage by the components interacting with the FHE Encryption Manager:

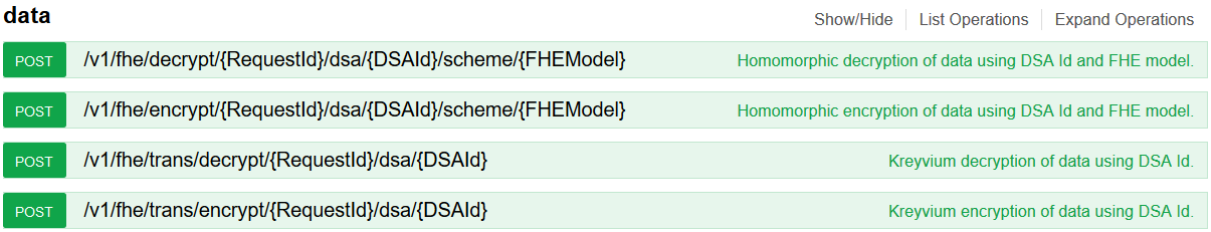


Figure 63: FHE Encryption Manager API from Swagger UI

Multiple encryption schemes in CTI data for FHE analysis purpose

According to blacklist verification use case, before encrypting entirely a CTI data and storing into the DPOS, we use an additional level of Kreyvium encryption for the fields that contain IPv4 addresses. Once those are encrypted, the CTI data can be encrypted by the DPO Encryption Manager. This encrypted IP will be transcrypted into FHE for analysis later.

8.3. Secure Audit Manager

All the C3ISP components should be integrated with the **Secure Audit Manager** to track all the critical operations (e.g. the policies evaluation and their enforcement results, analytics execution, etc.) in order to achieve the accountability requirement for the C3ISP Framework.

We plan to have two distinct features to support the auditing process:

- Inter-component auditing: trace the REST calls between the modules, components and subsystems. This helps assuring the right workflow is in place and seeing at each moment who is calling what;
- Internal-component auditing: audit critical events raised by a module or a component (e.g. policy evaluation, user attribute fetch, DPO bundle encryption/decryption, etc.). This assure accountability of actions for relevant software artefacts.

By combining both features, we can obtain a complete picture of what the C3ISP Framework is doing and ultimately provide an accountability track that can be useful to demonstrate that the Data Sharing Agreement is being respected.

Figure 64 describes the architecture that supports the above-mentioned features:

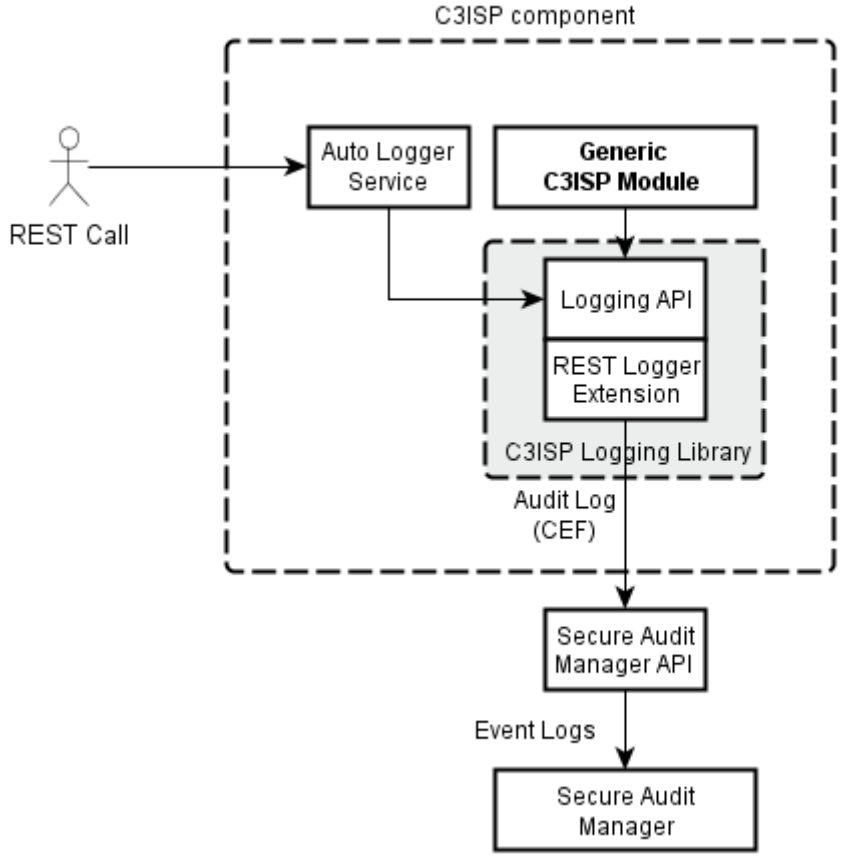


Figure 64: Secure Audit Manager architecture

The "REST call" is any request to a C3ISP component: since the C3ISP Framework employs a REST-based micro-services architecture, all the communication between components and modules happens via RESTful web services (inter-component auditing). The "Auto Logger Service" is a component that automatically traces received REST requests and traces them via a "C3ISP Logging Library". It is that library that feeds a Secure Audit Manager service. In order to accommodate for different Audit Managers, we interpose a Secure Audit Manager API that abstracts the interface between the C3ISP Logging Library and the Audit Manager. The Auto Logger Service will provide a first basic level of logging, with common attributes and values for all the components (e.g. timestamp, source address, target address, module involved, REST request parameters, etc.).

The "Generic C3ISP Module" is any of the C3ISP modules in the C3ISP architecture: to support internal-component auditing, also the C3ISP modules will be able to trace their activity by integrating with the C3ISP Logging Library. Also, this logging will be traced in the Secure Audit Manager. The logging from the C3ISP modules will contain more detail than those from the Auto Logger Service, since each component owner will be able to set precisely the logging attributes and values.

8.3.1. Implementation and Integration Status

At M24 the Secure Audit Manager has not been integrated yet and will be part of the next cycle of activities. However, we have a prototype of it that is running in isolation.

The C3ISP Logging Library is based on the REST paradigm: we used a standard Logging API (i.e. the SLF4J[22] logging framework) and extended it with a RESTful extension that is able to send logs via web services requests. At the end, also the logging mechanism itself could be traced by the Auto Logger Service.

Since all the C3ISP components are deployed on a servlet container (Apache Tomcat), the Auto Logger Service can act as a global Servlet filter that intercepts all the HTTP requests/responses and traces their details.

The Secure Audit Manager API is a REST-based server that accepts the REST calls from the C3ISP Logging Library and will send them to the Audit Manager. In fact, the Secure Audit Manager will be an off-the-shelf product: we plan to use CEF logging format, in such a way that we will be able to integrate with many products, both OSS and commercials.

The major goals for the final prototype are to deploy the Auto Logger Service on all the C3ISP subsystems and integrate the C3ISP Logging Library into the C3ISP components. Also, a Secure Audit Manager product will be installed. We currently are evaluating the open source Graylog[23] product, which will probably meet our needs as a reference auditing platform.

[22] Simple Logging Facade for Java, https://www.slf4j.org/

[23] Graylog, https://www.graylog.org/

9. Updated Data Flow Diagrams

This section illustrates the updated data flows by describing the revised UML sequence diagrams implemented for the C3ISP Framework at M24. We did also a name change, where all the *actionCTI* has been renamed to *actionDPO* (e.g. *createCTI* is now *createDPO*), to make it clearer that they work on Data Protected Object, which is the bundle of the CTI with the corresponding DSA.

9.1. Create DPO

This flow describes the operation that an external entity issues when it wants to provide input data to the C3ISP Framework. With respect to D7.2, we collapsed two operations (the creation for raw data and the one for normalised data), by allowing a parameter in the API to specify the corresponding scenario.

The Create DPO (*createDPO* method) is provided by the ISI API component and its sequence diagram is reported next:

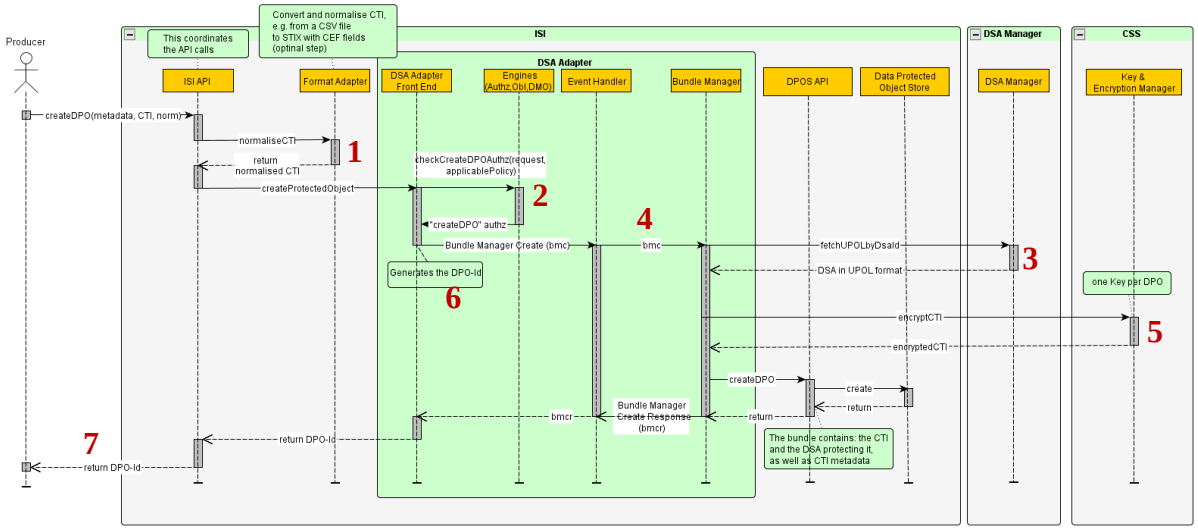


Figure 65: Create DPO Sequence Diagram

The diagram expresses greater details respect to the previous D7.2 description. In particular, we can see that the ISI API handles the acquisition of both raw and normalised CTI data (*norm* flag in the call[24]): raw CTI data has to be appropriately formatted before feeding it into C3ISP (hence the interaction with the Format Adapter) to encode the data in STIX format and, in case of log data, to format it in CEF, if necessary (**1**). Once the CTI data is in the proper format, the ISI checks if the *createDPO* operation is permitted (per the DSA policies) for the requestor component (**2**), then contacts the DSA Adapter for the bundle creation. The Bundle Manager fetches the UPOL policy (**3**) corresponding to the specified DSA (*metadata* parameter in the *createDPO* contains a reference to the DSA Id to be used). The Bundle Manager is coordinated by the Event Handler (see changes described in 5.1) and so there are messages exchanges with it: in fact, the DSA Adapter Front End sends a message "Bundle Manager Create" (bmc) that kicks off the Bundle Manager activities (**4**). Then the Bundle Manager interacts with the Key and Encryption Manager (part of the CSS subsystem) to create the encrypted bundle (**5**) that we call DPO – Data Protected Object (we have one key per DPO, stored in a vault, see Section 5.1.6). It is interesting to show that the identifier of the DPO that will be created is defined by

[24] In the final implementation of the Format Adapter, it will be able to automatically detect if the CTI data has to be specifically formatted or not. See also 5.2.

the DSA Adapter Front End (**6**). The DPO-Id is finally returned as a result parameter of the *createDPO* API (**7**).

9.2. Read DPO

This flow describes the operation that an external entity issues when it wants to get a data from the C3ISP Framework: either a previously submitted data, or a shared data by some other entity being subject to the defined DSA policies evaluation. Like the *createDPO* operation, with respect to D7.2, the following sequence diagram shows more details about the *readDPO* method, as a result of the developed integration with the Event Handler publish-subscribe mechanism (see Section 5.1.2).

The Read DPO is provided by the ISI API component and its sequence diagram is reported next:

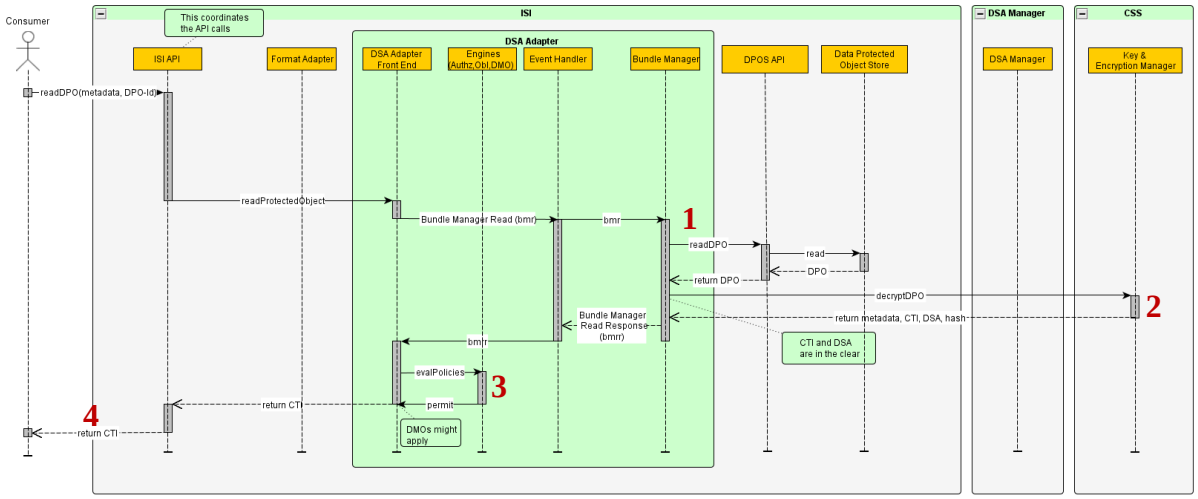


Figure 66: Read DPO Sequence Diagram

The main differences are the interaction between the Bundle Manager and the Event Handler, and the fact that now the DSA policies are bundled within the DPO (and so it is not required to fetch them from the DSA Manager). The DSA Adapter kicks off the Bundle Manager flow to retrieve the DPO bundle (**1**); the Bundle Manager then decrypts the DPO to extract the CTI and the DSA in the clear (**2**). Then there is the DSA policy evaluation (including DMOs, if applicable) (**3**) before returning the CTI in the clear to the caller (**4**).

9.3. Delete DPO

This flow describes the operation that an external entity issues when it wants to delete a DPO from the C3ISP Framework.

The sequence diagram has been revised and now the *deleteDPO* uses a *readDPO* first (by issuing a "bmr" message (**1**) to the Event Handler), as reported in the next figure.

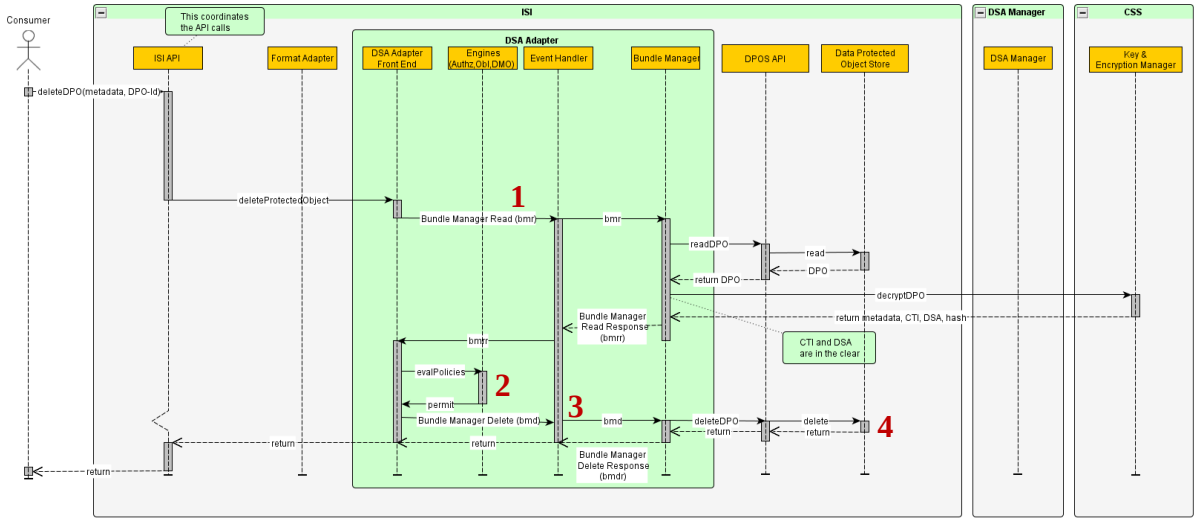


Figure 67: Delete DPO Sequence Diagram

Once the DSA policies allows the delete operation (**2**), a Bundle Manager Delete "bmd" message is sent to the Event Handler (**3**), which performs the delete against the DPOS (**4**).

9.4. Invoke C3ISP analytics service

This flow describes the feature provided by the C3ISP Framework for invoking a generic analytics service on a DPO (or list of DPOs), given the service name and the DPO-Id (or DPO-Ids).

The following diagram shows the updated sequence diagram:

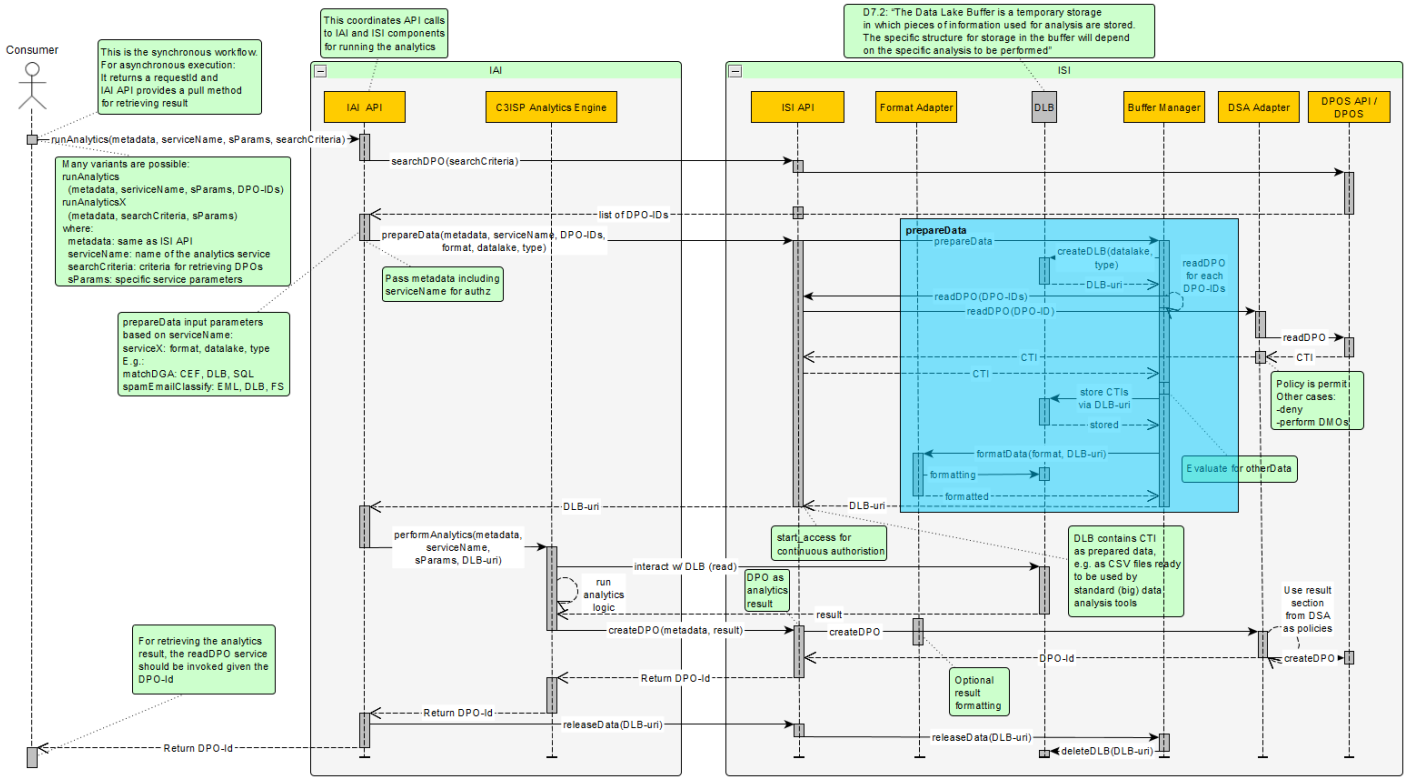


Figure 68: Invoke Analytics Service Sequence Diagram

This workflow has undergone through a quite extensive activity to detail it; however, the original flow from D7.2 is still the basis, even if it was a simplified version.

In particular, the following changes have to be highlighted:

- The *runAnalytics* signature allows the specification of "searchCriteria" to find out the DPO-IDs where the analytics will run over. The criteria are specified by using the same conventions of the *SearchDPO* API described in Section 5.4. Additionally, the *runAnalytics* can specify directly the list of DPO-IDs or be implemented for each analytics service (*runAnalytics**X***, e.g. runAnalyticsSearchDGA);
- The *prepareData* workflow has been expanded (see the blue box). It is implemented by the Buffer Manager component (see Section 5.3) and is able to create a "data lake" properly formatted for the analytics, by evaluating the policies attached to each DPO that will be part of the analysis;
- The *performAnalytics* works on the prepared "data lake";
- When the analytics is over, it submits the result CTI to the C3ISP Framework via a *createDPO* (see 9.1) in such a way that it becomes new data for the system. The DSA has a dedicated section that can contain the specific DSA policies to apply to the results (i.e. derived objects, see 7.1);
- Once the analytics has completed its job, the temporary data lake can be released via the releaseData API of the Buffer Manager (it is finally deleted, see 5.3).

As a final note, to keep the diagram to a reasonable simplification level, the DPOS API and DPOS has been collapsed in a single node, but in reality they are independent components.

9.5. Invoke legacy analytics service

Invoking legacy analytics works much the same as invoking a C3ISP analytics service (see Section 9.4). The legacy analytics service name will be specified as parameter in the *runAnalytics* service call. The IAI API will trigger the *prepareData* workflow (implemented by the Buffer Manager) to create a Virtual Data Lake instance and populate it with the requested CTIs (identified by the DPO-IDs). The VDL-uri (the equivalence of DLB-uri depicted in Figure 68) will be returned to the IAI API, which in turn will pass it to the legacy analytics engine; this will be supported by specific software component of the Pilot that makes use of the legacy analytics engine. The legacy analytics engine will then access the VDL directly; the analytics result will not be submitted to the C3ISP Framework, since legacy analytics do not have the awareness of this possibility and cannot be updated for that.

10. Updates on the Development and Test Bed Environments

In the following sections, we describe what has currently been updated or added, in terms of base hardware/operating system settings and software configuration, to Development and Test Bed Environments with respect to deliverable D7.2 (section 10) to match the requirements reported in deliverable D7.1 (sections 3.1 and 3.2).

10.1. Development Environment

10.1.1. Base configuration

There are no changes to the configuration of the VM that hosts the Development environment.

10.1.2. Software configuration

This section reports the status of the development tools mentioned in deliverable D7.1.

The new installed tools are the following:

- **Nexus Artifact Repository**[25] is a centralized repository to manage components from dev through delivery (binaries, containers, assemblies, and finished goods), build artefacts, and release candidates in one central location. It supports in particular requirement [C3ISP-Dev-001]. Nexus, in development environment, is integrated with Jenkins and Maven. We use it mainly to share common C3ISP artefacts that are used cross-modules (see next figure). Its access control mechanism has also been integrated for user authentication and authorisation with the C3ISP OpenLDAP service. The configured service is available at https://devc3isp.iit.cnr.it/nexus/.

A sample screenshot is provided in the next figure:

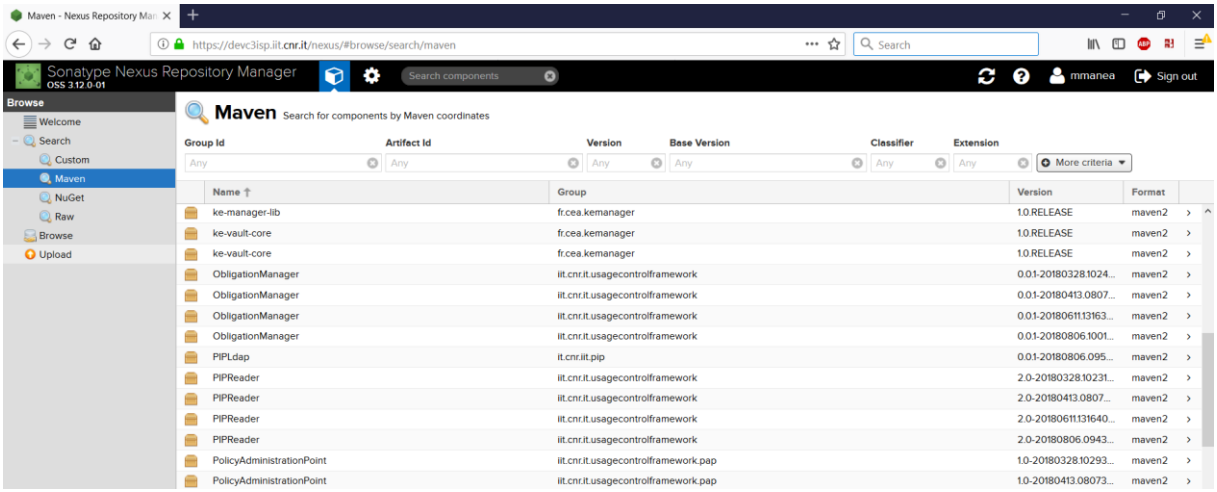


Figure 69: Nexus Web Interface

- **Trac**[26] is a system for bug/issue tracking and management. It addresses in particular requirement [C3ISP-Dev-004]. We defined on Trac all the C3ISP components and modules, as shown in Figure 70. The configured service is available at https://devc3isp.iit.cnr.it/trac/.

[25] https://www.sonatype.com/nexus-repository-oss

[26] http://trac.edgewall.org/

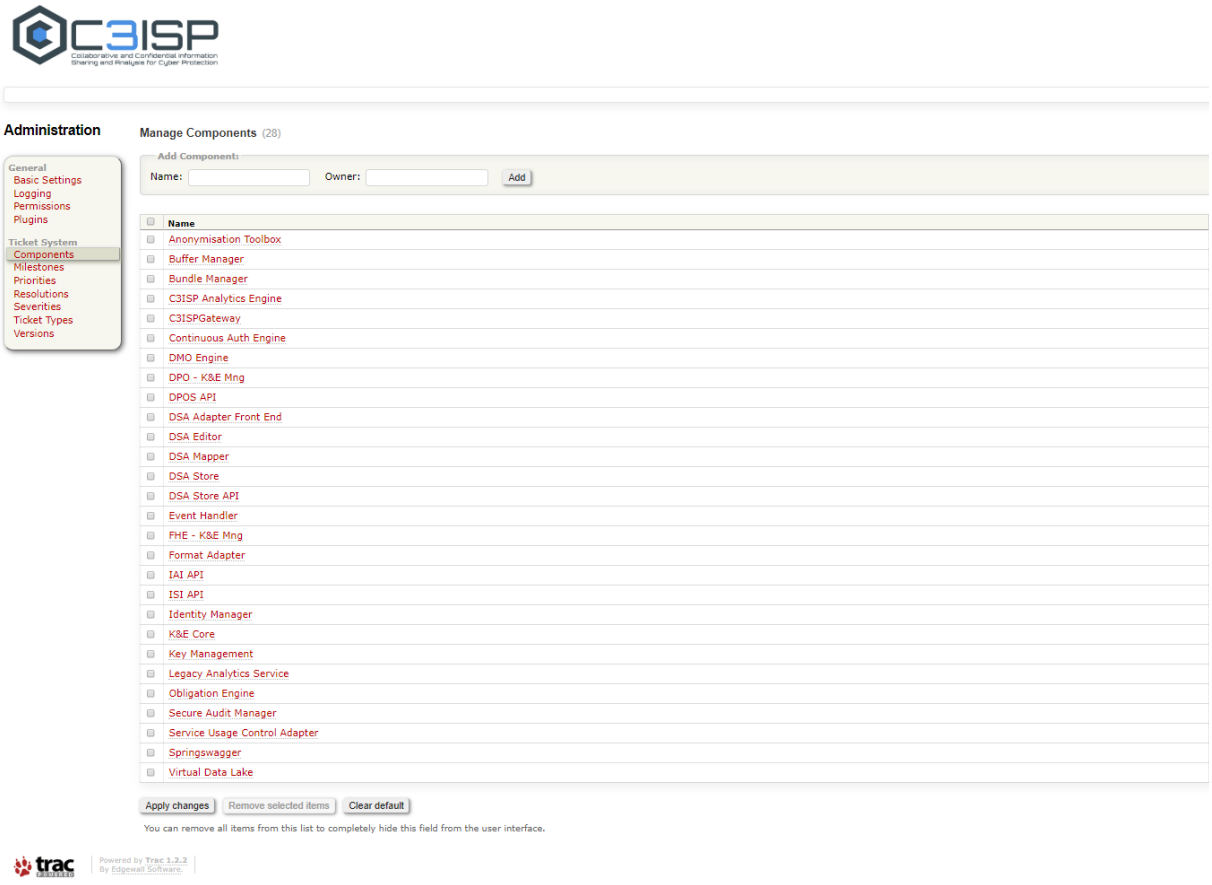


Figure 70: Trac C3ISP modules list

We also started using Trac as an internal wiki tool, where we publish technical material regarding the implementation. Next figure shows a sample:

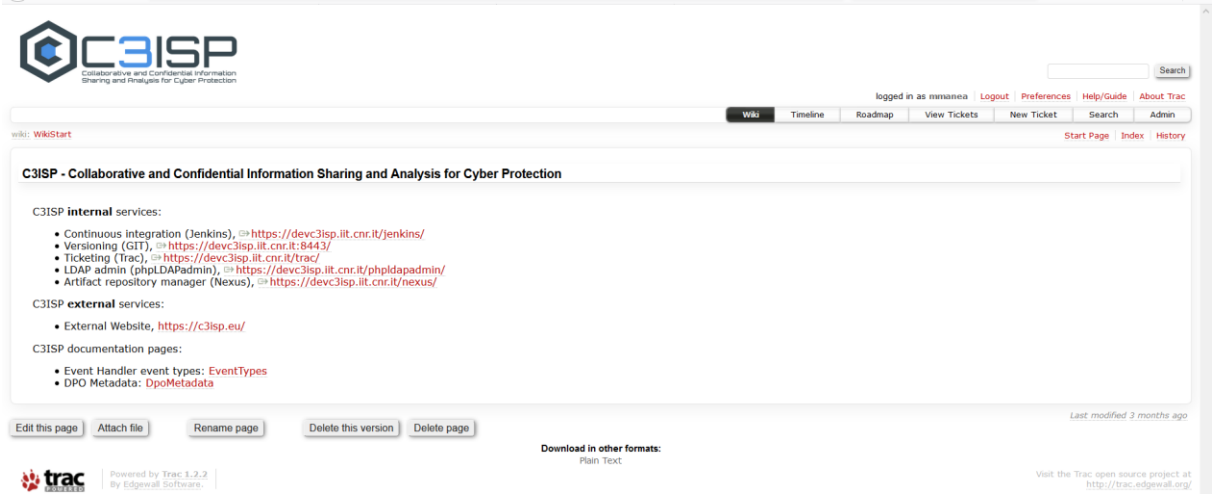


Figure 71: Trac Wiki pages

10.2. Test Bed Environment

10.2.1. Base configuration

The configuration for each machine of the test bed environment is illustrated on Table 39 under the *Specifications* column. In addition, all machines are equipped with Ubuntu 16.04 LTS as operating system.

Table 39 – C3ISP subsystems on Test Bed Environment

Subsystem	Virtual Machine	Specifications	Notes
DSA Manager	dsamgrc3isp.iit.cnr.it	vCPU = 2 Ram = 4GB Storage = 40GB	This virtual machine hosts the DSA Manager subsystem.
ISI	isic3isp.iit.cnr.it	vCPU = 2 Ram = 6GB Storage = 100GB	This virtual machine hosts the Information Sharing Infrastructure subsystem.
IAI	iaic3isp.iit.cnr.it	Cores = 40 Ram = 256GB Storage = 6TB	This server hosts the Information Analytics Infrastructure subsystem. We decided to install this subcomponent on a physical server with high performance due to the high load, for instance the homomorphic encryption analytics.
CSS	kec3isp.iit.cnr.it	vCPU = 8 Ram = 6GB Storage = 20GB	This virtual machine hosts at M24 the Key and Encryption Manager component. We plan to install shortly also the Secure Audit Manager component (off-the-shelf-product).
MISP	mispc3isp.iit.cnr.it	vCPU = 2 Ram = 4GB Storage = 20GB	This virtual machine hosts the components that will be used to interface with the Open Source Threat Intelligence Platform & Open Standards for Threat Information Sharing.
ISP Pilot	ispc3isp.iit.cnr.it	vCPU = 4 Ram = 4GB Storage = 60GB	This virtual machine hosts all services and components for the ISP Pilot that at M24 are used to interact with the C3ISP Framework, and in particular with the IAI. It hosts the local ISI due to the C3ISP Hybrid deployment model used by the pilot.

CERT Pilot	90.147.82.10	vCPU = 4 Ram = 4GB Storage = 60GB	This virtual machine hosts all services and components for the CERT Pilot that at M24 are used to interact with the C3ISP Framework. It hosts the local ISI due to the C3ISP Hybrid deployment model used by the pilot.
Enterprise Pilot	entc3isp.iit.cnr.it	vCPU = 4 Ram = 4GB Storage = 60GB	This virtual machine hosts all services and components for the Enterprise Pilot that at M24 are used to interact 2with the C3ISP Framework, and in particular with the IAI.
SME Pilot	smec3isp.iit.cnr.it	vCPU = 4 Ram = 4GB Storage = 60GB	This virtual machine hosts all services and components for the SME Pilot that at M24 are used to interact with the C3ISP Framework. It hosts the local ISI due to the C3ISP Hybrid deployment model used by the pilot.

10.2.2. Software configuration

The new installed tools are the following:

- **MySQL**[27] is a Relational database management system. It is used for:
  - Managing users and DSAs for the DSA Editor (Profile Store);
  - Implementing VDL for some legacy analytics engines;
  - Storing keys for HashiCorp Vault, used by K&E Manager;
  - Backend for DGA analytics (see D8.3);
- **MongoDB**[28] is a non-relational (No-SQL) document-oriented database for storing data in flexible, JSON-like documents. It is used for:
  - Store DPO metadata to be used together with the DPOS;

[27] https://www.mysql.com/

[28] https://www.mongodb.com/

  - DSA Store;
- **Apache Hadoop Distributed File System** (**HDFS**)[29] is a distributed Java-based file system for storing large volumes of data across multiple machines. It is used for:
  - Storing VDL for legacy analytics services;
  - Storing Data Lake Buffers (created by the Buffer Manager) for C3ISP analytics services;
  - DPOS.

Table 3 contains also software version details.

[29] https://hadoop.apache.org/

11. Conclusions

This deliverable has described the implemented first version of the C3ISP Framework. We are continuously tuning the implementation by looking at the Pilots integration activities. In fact, the validation phase that will conclude at M26 is an important milestone also to verify that the implemented architecture is completely sound and meets correctly the Pilot requirements and needs. We will use the outcomes of that activity, to refine the components developments, fix defects and improve the security and usability of the C3ISP Framework.

In fact, we have some activities already on the short-term roadmap, like the activation of the Service Usage Control Adapter and the setup of the Secure Audit Manager, which will provide additional security measures to the C3ISP Framework. We also need to work on a stricter integration with the Identity Manager component, in particular regarding the user authentication and new attributes for policy evaluation, in order to meet the requirements of the DSA rules that will be refined by the Pilots. Further, as the analytics services will improve and new services will be possibly added, we will work on improving the performances of the C3ISP Framework, should any bottleneck arises during the Pilot testing and integration activities. Testing of the various deployment models, physically distributed and hybrid will increase confidence in detecting performance results.

12. Appendix 1: Swagger API URLs

The following tables shows the published URLs for inspecting the API signature as provided by Swagger:

Table 40 – Swagger URLs

Subsystem	Component	Module	URL
ISI	DSA Adapter	DSA Adapter Front End	https://isic3isp.iit.cnr.it/dsa-adapter-frontend/swagger-ui.html
ISI	DSA Adapter	Event Handler	https://isic3isp.iit.cnr.it/event-handler/swagger-ui.html
ISI	DSA Adapter	Continuous Authorization Engine	https://isic3isp.iit.cnr.it/UsageControlFramework/swagger-ui.html
ISI	DSA Adapter	Continuous Authorization Engine/Multi Resource Handler	https://isic3isp.iit.cnr.it/multi-resource-handler/swagger-ui.html
ISI	DSA Adapter	Obligation Engine	https://isic3isp.iit.cnr.it/trigger-engine/swagger-ui.html https://isic3isp.iit.cnr.it/action-engine/swagger-ui.html
ISI	DSA Adapter	DMO Engine	https://isic3isp.iit.cnr.it/dmo-engine/swagger-ui.html
ISI	DSA Adapter	Bundle Manager	https://isic3isp.iit.cnr.it/bundle-manager/swagger-ui.html
ISI	Format Adapter	-	https://isic3isp.iit.cnr.it:9443/format-adapter/api-docs/
ISI	Buffer Manager	-	https://isic3isp.iit.cnr.it/buffer-manager/swagger-ui.html
ISI	DPOS API	-	https://isic3isp.iit.cnr.it/dpos-api/swagger-ui.html
ISI	API	-	https://isic3isp.iit.cnr.it/isi-api/swagger-ui.html
IAI	IAI API	-	https://iaic3isp.iit.cnr.it/iai-api/swagger-ui.html
IAI	C3ISP Analytics Engine	FHE Analytics	https://iaic3isp.iit.cnr.it/fhe-conn-malicious-host/swagger-ui.html
DSA Manager	DSA Editor	-	Web GUI: https://dsamgrc3isp.iit.cnr.it/DSAEditor/
DSA Manager	DSA Mapper	-	https://dsamgrc3isp.iit.cnr.it:8443/dsa-mapper/swagger-ui.html
DSA Manager	DSA Manager Gateway	-	https://dsamgrc3isp.iit.cnr.it/DSAAPI/swagger-ui.html
DSA Manager	DSA Store API	-	https://dsamgrc3isp.iit.cnr.it/dsa-store-api/swagger-ui.html
CSS	Key and Encryption Manager	K&E Core	https://kec3isp.iit.cnr.it/ke-core-manager-api/swagger-ui.html

CSS	Key and Encryption Manager	DPO Key Manager	https://kec3isp.iit.cnr.it/dpos-key/swagger-ui.html
CSS	Key and Encryption Manager	FHE Key Manager	https://kec3isp.iit.cnr.it/fhe-keys/swagger-ui.html
CSS	Key and Encryption Manager	DPO Encryption Manager	https://kec3isp.iit.cnr.it/dpos-encryption/swagger-ui.html
CSS	Key and Encryption Manager	FHE Encryption Manager	https://kec3isp.iit.cnr.it/fhe-encryption/swagger-ui.html

13. References

This section lists the references used throughout the document:

[1] RESTful services: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, retrieved on Sep 11th, 2018

[2] Swagger: http://swagger.io, retrieved on Sep 11th, 2018

[3] Swagger UI: http://swagger.io/swagger-ui, retrieved on Sep 11th, 2018

[4] Swagger Editor: https://editor.swagger.io/, retrieved on Sep 11th, 2018

[5] YAML data serialisation standard: http://yaml.org/spec/1.2/spec.html, retrieved on Sep 11th, 2018

[6] Springfox: http://springfox.github.io/springfox/docs/current, retrieved on Sep 11th, 2018

[7] Spring Boot: http://projects.spring.io/spring-boot, retrieved on Sep 11th, 2018

[8] JAX-RS: https://github.com/jax-rs, retrieved on Sep 11th, 2018

[9] Clements, Paul, and Linda Northrop. Software product lines: practices and patterns. Vol. 3. Reading: Addison-Wesley, 2002

[10] Carpov, Sergiu, Paul Dubrulle, and Renaud Sirdey. "Armadillo: a compilation chain for privacy preserving applications." Proceedings of the 3rd International Workshop on Security in Cloud Computing. ACM, 2015

[11] Fan, Junfeng, and Frederik Vercauteren. "Somewhat Practical Fully Homomorphic Encryption." IACR Cryptology ePrint Archive 2012 (2012): 144

[12] Brakerski, Zvika. "Fully homomorphic encryption without modulus switching from classical GapSVP." Advances in cryptology–crypto 2012. Springer, Berlin, Heidelberg, 2012. 868-886

[13] Gentry, Craig, Shai Halevi, and Nigel P. Smart. "Fully homomorphic encryption with polylog overhead." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2012

[14] Bajard, Jean-Claude, et al. "Efficient reductions in cyclotomic rings-Application to Ring-LWE based FHE schemes." International Conference on Selected Areas in Cryptography. Springer, Cham, 2017

[15] Jäschke, Angela, and Frederik Armknecht. "Field Work: Choosing the Best Encoding of Numbers for FHE Computation." Proc. of CANS. 2017

[16] Chillotti, Ilaria, et al. "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds." International Conference on the Theory and Application of Cryptology and Information Security. Springer, Berlin, Heidelberg, 2016

[17] Chillotti, Ilaria, et al. Improving TFHE: faster packed homomorphic operations and efficient circuit bootstrapping. Cryptology ePrint Archive, Report 2017/430, 2017

[18] Chillotti, Ilaria, et al. "Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE." International Conference on the Theory and Application of Cryptology and Information Security. Springer, Cham, 2017

[19] Gentry, Craig, Amit Sahai, and Brent Waters. "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based." Advances in Cryptology–CRYPTO 2013. Springer, Berlin, Heidelberg, 2013. 75-92

[20] Alperin-Sheriff, Jacob, and Chris Peikert. "Faster bootstrapping with polynomial error." International Cryptology Conference. Springer, Berlin, Heidelberg, 2014

[21] Ducas, Léo, and Daniele Micciancio. "FHEW: bootstrapping homomorphic encryption in less than a second." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2015

[22] Jaehong Park, R. S. (2002). Towards usage control models: beyond traditional access control. SA CMA T, (pp. 57 - 64)

[23] RFC 6749, OAuth2: https://tools.ietf.org/html/rfc6749, http://oauth.net/2/, retrieved on Sep 11th, 2018

[24] SPID – https://www.spid.gov.it, retrieved on Sep 11th, 2018