D7.2

First version of C3ISP Architecture

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

C3ISP

Collaborative and Confidential Information Sharing and Analysis for Cyber Protection

Due date of deliverable: 30/09/2017
Actual submission date: 30/09/2017

30/09/2017

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: C. Gambardella, M. Manea (HPE), T. Nguyen, V. Herbert (CEA), I. Herwono (BT), R. de Lemos, D. Chadwick (UNIKENT), F. Di Cerbo, J. Boehler (SAP), P. Mori, A. Saracino, G. Costantino, I. Matteucci (CNR), J. Dobos (3DREPO)

Approved by: R. de Lemos (UNIKENT), F. Di Cerbo (SAP)

Revision History

Version	Date	Name	Partner	Sections Affected / Comments
0.1	26-Jun-2017	M. Manea	HPE	Initial ToC
0.2	03-Jul-2017	M. Manea	HPE	Appointed UNIKENT and SAP as internal reviewers
0.3	25-Jul-2017	M. Manea, C. Gambardella	HPE	Added HPE contribution
0.4	28-Jul-2017	C. Gambardella, R. de Lemos, D. Chadwick, I. Herwono, T. Nguyen, V. Herbert	HPE, UNIKENT, BT CEA	Merged contribution from UNIKENT, BT, CEA
0.5	22-Aug-2017	P. Mori, A. Saracino, G. Costantino, I. Matteucci	CNR	Merged contribution from CNR
0.6	28-Aug-2017	J. Dobos	3DREPO	Merged contribution from 3DREPO
0.7	30-Aug-2017	F. Di Cerbo	SAP	Merged contribution from SAP
0.8	08-Sep-2017	M. Manea, C. Gambardella	HPE	Ready for Internal Review
0.9	26-Sep-17	J. Ziembicka, R. de Lemos	UNIKENT	Internal review
1.0	29-Sep-17	C.Gambardella, M.Manea	HPE	Final version

Executive Summary

The objective of this deliverable is to define first version of the reference architecture for C3ISP Framework. The starting point for this work has been the requirements collected and described in the deliverable 7.1, in particular those involving the project Pilots. The architecture meets the functional and non-functional requirements derived from the Pilot's business needs and it provides a generic Framework that serves as basis for the following development phase that will lead to the first version of the C3ISP Framework reference architecture implementation (due at Month 24). The content of the deliverable has been built in collaboration with our partners, taking into consideration the needs of the other work packages (in particular WP6, representing the Pilots, and WP8), by collecting and considering different viewpoints in order to realise a coherent, flexible and easily integrated architecture.

Table of contents

1. Introduction

1.1. Overview

This document presents the first version of the reference architecture for the C3ISP Framework, as it has been designed to satisfy the requirements from the project Pilots. This reference architecture will form the basis under which the development and integration activities will take place during the next months, while implementing the first version of the C3ISP reference architecture by Month 24 (see DoW). That implementation will be installed in the C3ISP test environment and will be used for project Pilots testing and integration tasks.

1.2. Deliverable Structure

The document is structured as follows:

- The description of the high-level architecture has been provided (Sect. 2), proposing the possible deployment models for it (Sect. 3).
- Sect. 4 is dedicated to the presentation of the instantiation of the architecture on the Pilots.
- Then, the deliverable provides a clear description of all the architectural components, with a dedicated section describing their details (Sect. 5-8).
- In order to provide a coherent view of the roles of all the subsystems, a set of data flow diagrams that describe the main operations of the reference C3ISP Framework (Sect. 9).
- Sect. 10 maps the requirements listed in the D7.1 to the C3ISP Framework components and functionalities of the reference architecture, in order to match the coverage in the designed architecture.
- The deliverable concludes by illustrating the status of the development and test bed environment at Month 12.

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 Storage
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

This section describes at a high-level the C3ISP Framework reference architecture (notion introduced in D7.1, Section 1.2), in particular the system actors, the set of subsystems with their core components as well as their interaction. The next sections will drill down and illustrate the design-level specification for each component. We have adopted the "Fundamental modeling concepts" (FMC) Framework [66] for describing the architecture components and the communication between them, realising the architectural diagrams in a top-down approach (from the high level architecture to the detailed subsystems).

The system actors are Prosumers entities, sometimes also expanded with their specific roles of Producer and Consumer. The role of Producer is assumed by the one who supplies its own data to the C3ISP Framework for sharing it with other entities, which are Consumers of the data; the sharing is regulated by policies (i.e. a set of rules) defined on the data. The Prosumer is a generalisation of the Producer and Consumer roles (shown in Figure 1); namely, an actor may use the C3ISP Framework assuming both the role of Producer or Consumer, meaning that in a collaborative approach, s/he can provide data to the C3ISP Framework in order to improve the knowledge base shared between the other actors and also take advantage from it, retrieving data supplied by others or run analytics services on the whole dataset.

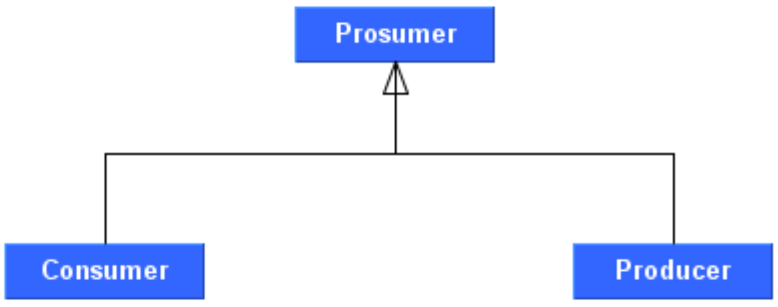


Figure 1: Actors hierarchy

Figure 2 shows the C3ISP reference architecture. From top-down, the C3ISP **subsystems** are:

- Data Sharing Agreement (DSA) Manager;
- Information Sharing Infrastructure (ISI);
- Information Analytics Infrastructure (IAI);
- Common Security Services (CSS).

The **Data Sharing Agreement (DSA) Manager** is in charge of handling the DSA object, which encapsulates the policy requirements (i.e. the set of rules) under which a protected data object (CTI data) can be used and shared. The DSA Manager subsystem handles the DSA lifecycle, from the editing phase to its usage till its termination. The Prosumers, collaboratively, define the sharing and analytics rules to be used by the C3ISP Framework to handle the Prosumers' provided data, thus considering all the set of jointly agreed requirements.

The **Information Sharing Infrastructure (ISI)** is the subsystem used by a Prosumer to provide data to the C3ISP users (i.e. the other Prosumers) under the governance of an appropriate DSA. Depending on several factors, like the trust assumptions a Prosumer has on the infrastructure, computational requirements, etc., the ISI can be both deployed locally and remotely, or remote-only. In both cases, the core feature of the subsystem is provided by the *DSA Adapter*, a component that is able to enforce the DSA rules, in particular those related to access and usage control, and the manipulation of the data itself (through the DMOs, Data Manipulation

Operations). A Producer is the actor that can submit its data to the C3ISP Framework and optionally a Consumer could use the ISI to retrieve shared data, both under the constraints of the DSA policies. Data protected under the DSA policies is stored securely (e.g. encrypted) in a *Data Protected Object Storage*.

The **Information Analytics Infrastructure (IAI)** subsystem provides the interface to invoke analytics services on the data that has been shared and (centrally) stored through the ISI. The analytics execution result is computed considering the associated DSA rules, which also apply to the handling of the resulting data that will have its own derived DSA rules (elaborated on the individual DSA each Prosumer assigned to their data). The result is submitted again to the ISI to be both shared between the C3ISP users and possibly used as an input for a new analytics service. In addition, to be able to manage legacy analytics engines (see requirement C3ISP-Fun-DA-011[1] from D7.1), upon such requests, the subsystem instantiates a *Virtual Data Lake*, prepared to be used by the required legacy analytics service: this lake[2] contains data that is prepared according to the DSA rules and usage constraints (e.g. part of the data could be anonymised, etc.). The Consumer actor is the person in charge of requesting the analytics services to be executed.

Finally, a bunch of integrated **Common Security Services (CSS)** are necessary to support the functions of the C3ISP Framework. For instance, access and usage control need identities and profile information from and *Identity Manager* to evaluate their logic; a *Secure Audit Manager* is necessary to trace the operations performed within the C3ISP Framework, in particular those related to access and usage decisions and in general to guarantee the system accountability to show it operates as planned and as specified in the DSA rules; a *Key and Encryption Manager* is necessary to provide the confidentiality of the computations (in the case of homomorphic encryption) and the secrecy for the shared CTI data.

[1] C3ISP-Fun-DA-011: "C3ISP provides an interface to integrate external analytics tools while preserving the policy compliance (i.e. extract data from C3ISP data lake and feed it into analytics tool)"

[2] Even though the Virtual Data Lake and the Data Protected Object Storage are both data repository, they serve different purposes: the former contains data ready for the analytics services processing, the latter contain protected *sharable* data (e.g. encrypted, as it will be clear in section 4.1.4).

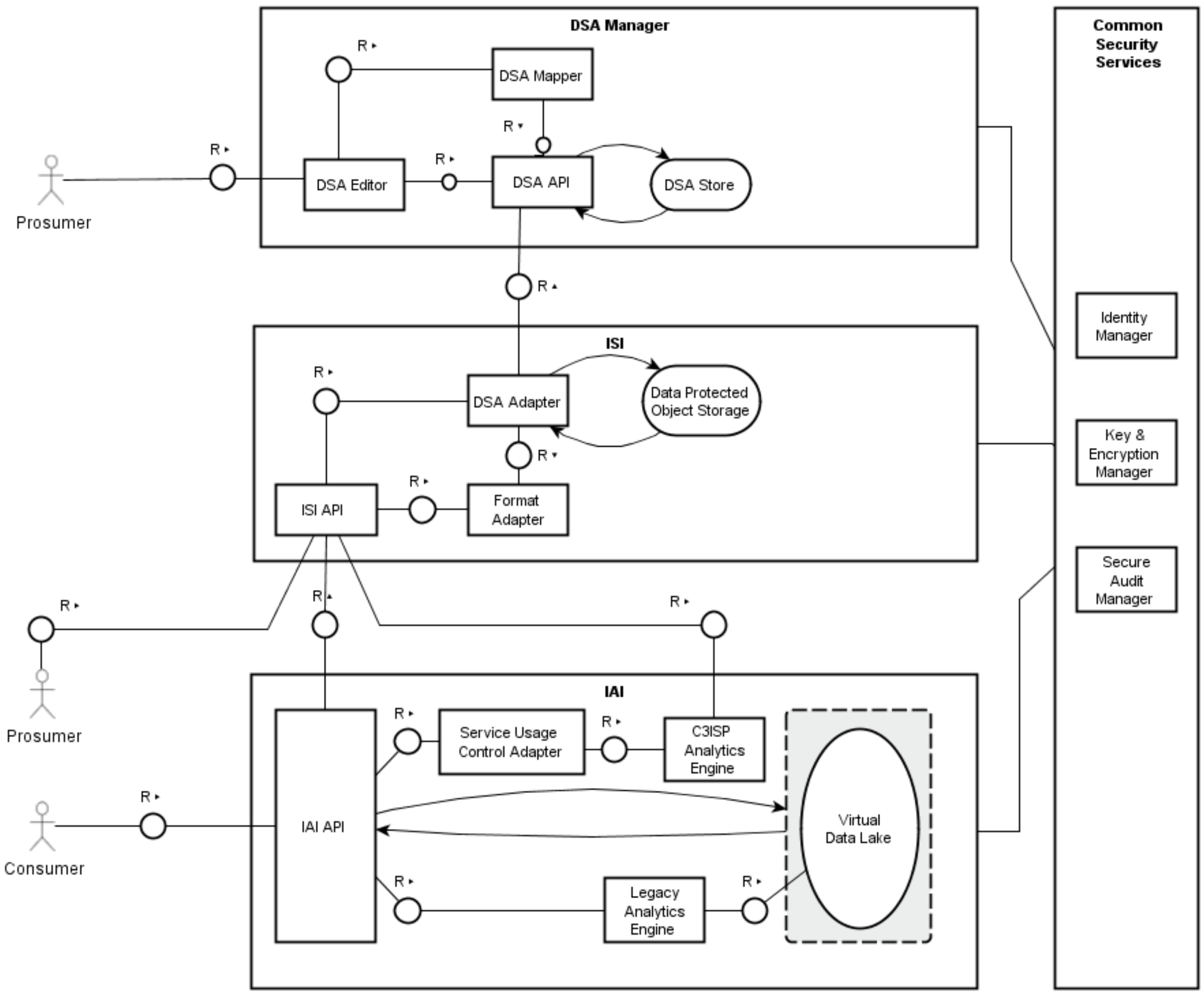


Figure 2: C3ISP high-level architecture – version 1 (Month 12)

2.1. *Micro-services architecture*

From a more technical standpoint, the approach we are following aims to realise a micro-services architecture that consider the framework not as a monolithic object but as a collection of small and mutually independent services, which can be implemented separately and communicate with others services through an exchange of messages (mainly using Web Services REST-based). The C3ISP Framework itself exposes API/services to the clients; these are intended not for end user but Pilot applications that will use these services to implement their business scenarios.

As typical in distributed architecture, all the communications happens via Web Services that need to be protected for authentication and authorisation, and we plan to leverage on the *Identity Manager* component for that (see Sect. 7.1). OAuth2 [57] is a well-known standard that can be used also for API protection. It is an authorisation framework that integrates with different authentication mechanisms. An important feature of OAuth2 is that it allows to propagate the identity of the calling user/client: in this way each component in the architecture will always be able to have the proper context of which Prosumer initiated the request.

In general C3ISP APIs have two levels of protection:

1. Authentication Level: API protection at http-level: this is used to guarantee access control at method level and to provide identity information to the API (through authentication);
2. Authorisation Level: API protection through DSA rules (e.g. for authorizing create/read/delete of CTI data): DSA could contain rules for granular method protection, like group-based authorization, RBAC or the more general ABAC model. For example, we could express that only users with a certain role can delete a shared file.

A micro-service architecture also fosters the development of a multi-tenanted C3ISP Framework, with the possibility to instantiate and scale each service easily, where C3ISP could be used by different tenants (i.e. Pilots) at the same time.

3. Deployment Models

We foresee different ways to implement the C3ISP Framework and we refer to them as deployment models. A deployment model is a specific C3ISP Framework configuration in which the C3ISP subsystems are deployed to match specific use case scenarios requirements. The Prosumers choose the deployment model according to their specific business requirements, in particular taking into account the level of trust they have or need for the use cases to be supported (each C3ISP Pilot evaluates and selects the most appropriate deployment model).

The deployment models describe where the main C3ISP subsystems can be deployed, specifically on-premises, in the Prosumer's environment, on a centralised environment or in a combination of these approaches.

We identified four deployment models for supporting a broad range of possible scenarios and the Pilots' implementation in particular:

- **Fully centralised**: both ISI and IAI are centralised;
- **Hybrid**: ISI is both on-premises and centralised, with a centralised IAI;
- **Distributed ISI**: ISI is on-premises only and IAI is centralised;
- **Fully distributed**: both ISI and IAI are on-premises.

The following picture shoes the trade-offs in the deployment models:

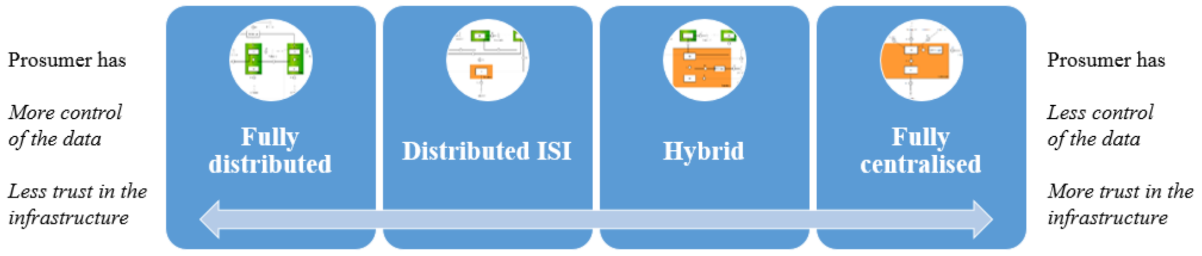


Figure 3: Deployment models trade-offs

A special consideration has to be made for the CSS subsystem: this subsystem has a critical role for the trustworthiness of the C3ISP Framework, in particular when considering distributed scenarios. For example, a distributed CSS for identity management could leverage on identity federation technologies. Key and encryption services could leverage on PKI to address key distribution issues. Auditing, however, should preferably be centralised, maybe at a third party, to address segregation of duties and non-repudiation. However, at this stage, we foresee a mainly centralised deployment for CSS with the assumption that it has to be trusted by all the Prosumers involved. In any case, the architecture can cope with existing consolidated solutions in the domain of identity management, key and encryption manager, auditing, provided that they use standard interfaces and in any way aligned with the expectations expressed here. Indeed, the project does plan to integrate free/open source software solutions to fulfil these needs (see section 7), leaving to the pilot implementation and deployment phases the task to verify the consistency and the security of the final system. In the upcoming deliverables, the results of the reference implementation activities will also deepen the descriptions and requirements for the security features of CSS services in distributed scenario, also providing possible implementation options.

The next sections describe each deployment model. For better identifying the trust boundaries in the deployment models, we have adopted a colour code in the components diagrams describing them. In particular, we use the *green* colour to delimit the trusted zones (where the

Prosumer has more control) and *orange* for the untrusted ones (where the Prosumer has less control).

3.1. Fully centralised: Centralised ISI and IAI

The simplest deployment model is an architecture where all the subsystems are installed in a centralised environment and the Prosumers operate on them remotely only (see Figure 4, where for the sake of simplicity CSS subsystem is not shown).

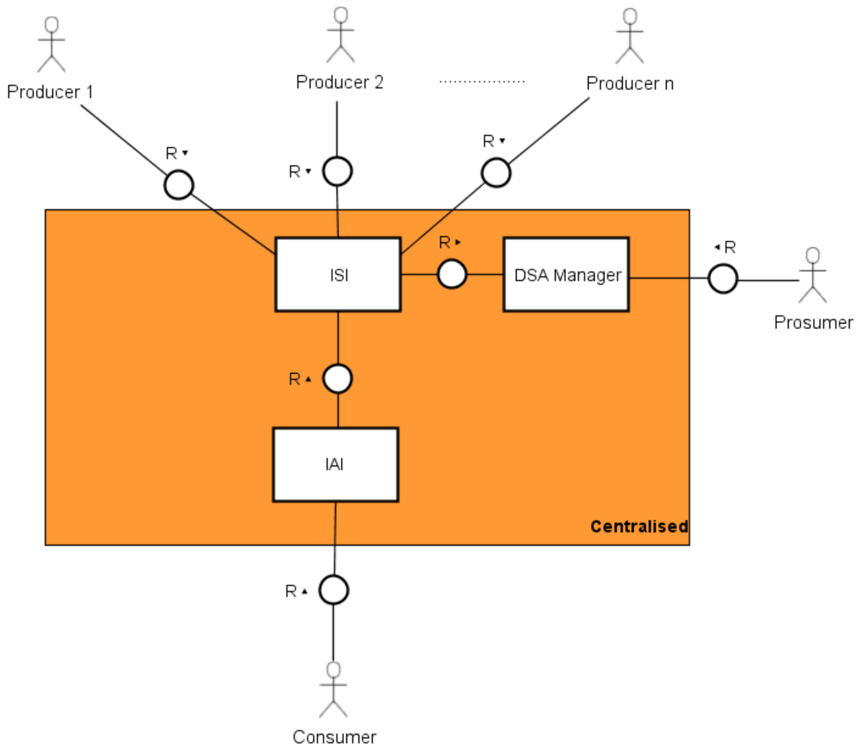


Figure 4: Fully centralised deployment model (Centralised ISI and IAI)

The picture shows the main C3ISP subsystems and focuses on the interactions between them and the Prosumers. The Producers (e.g. Producer 1, 2, n) submit data to the C3ISP Framework interacting with a centralised remote ISI: the sharing is regulated by policies defined in a DSA. The Prosumers define DSAs using the services provided by a DSA Manager (deployed with a SaaS approach) and the DSAs are stored in a common central repository. The Consumers use the analytics services exposed by a centralised IAI on the data provided by the ISI.

3.2. Hybrid: On-Premises ISI with Centralised ISI and IAI

According to some implementation preferences, like the level of trust required by the use case or particular business needs, the ISI can be deployed both on-premises, in the local environment of the Producer, ***and*** in a centralised infrastructure, as schematised in the following picture.

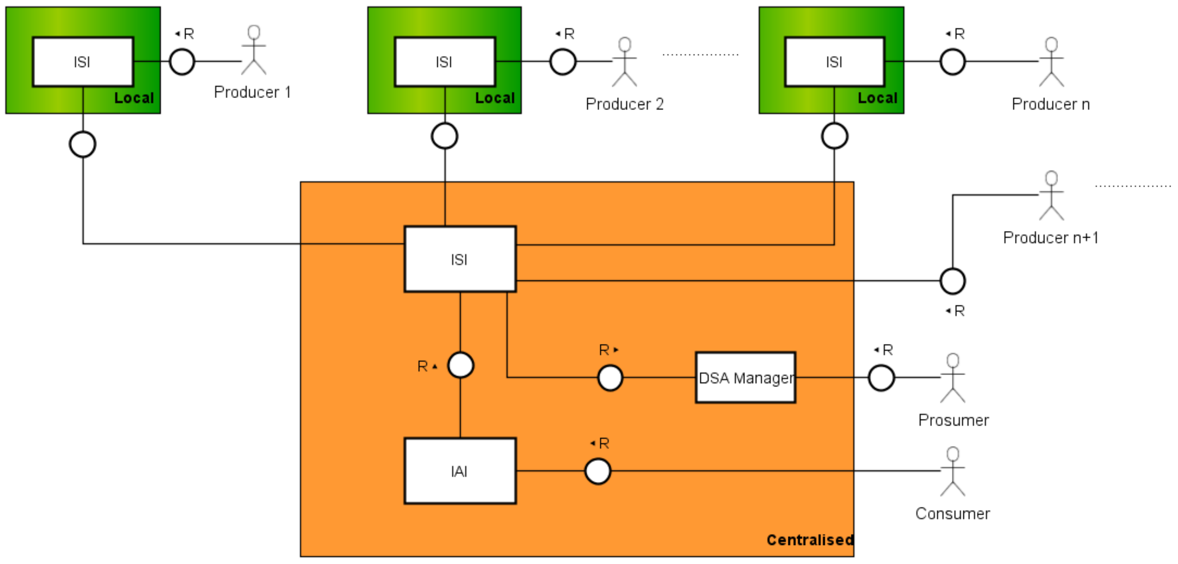


Figure 5: Hybrid deployment model (On-Premises ISI with Centralised ISI and IAI)

In this approach, a locally instantiated ISI allows a Producer (e.g. Producer 1) to share data with others Producers (e.g. Producer 2, Producer n), thanks to a centralised ISI, which is used to submit and retrieve the data to be shared with the C3ISP actors, as well as to interact with a centralised IAI for the analytics services. Some DSA policies could be enforced locally (on each local ISI) and some remotely: this model is well suited when a Producer wants to apply DMOs like data anonymisation (or some access/usage rules) before sharing its data with the centralised ISI infrastructure, where other specific DSA policies might apply. For example, a Producer could pre-process its data by anonymising some parts, before moving them outside of its premises (so out of its control) to be further shared and used with the analytics services.

This hybrid model also allows a producer (e.g. Producer n+1) to interact directly with the central ISI, thus making it a generalisation of the *Fully centralised* model described earlier.

3.3. *Distributed ISI: On-Premises only ISI and Centralised IAI*

If the Producers need to have full control of the ISI component and want to interact directly with the IAI, or have others constraints which do not allow a remote ISI (e.g. do not want to store data remotely because of luck of trust), an on-premises-only ISI deployment model is the best option (described in the next picture).

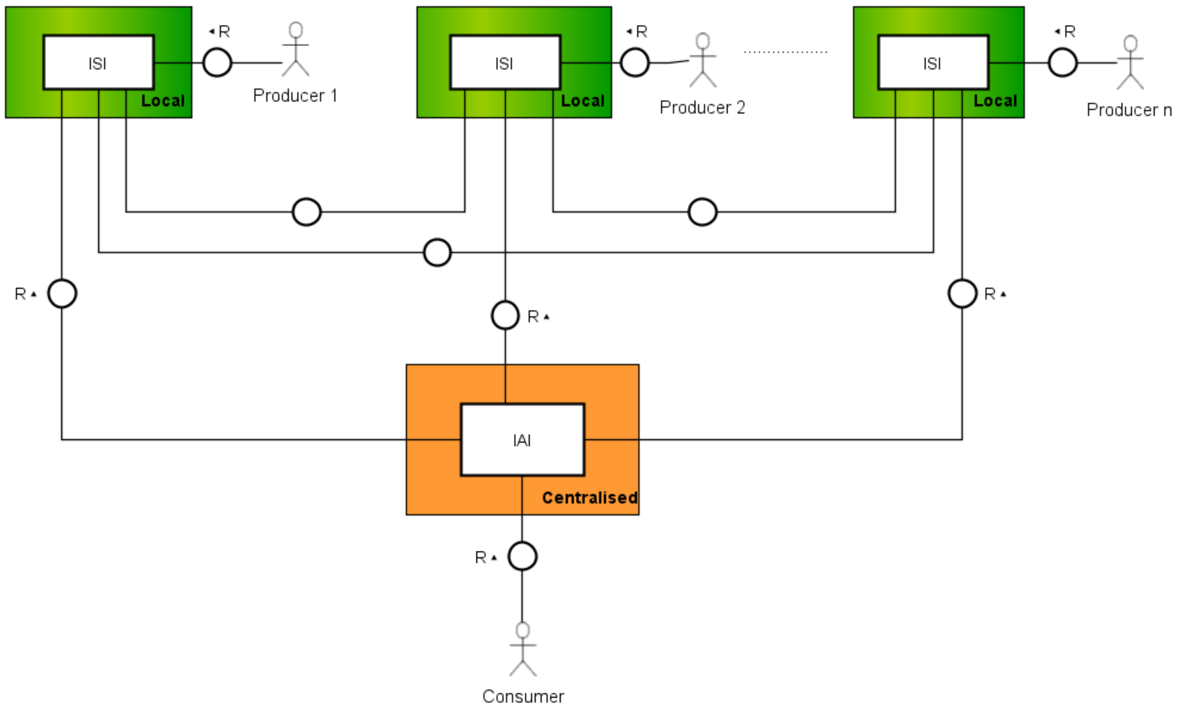


Figure 6: Distributed ISI deployment model (On-Premises only ISI and Centralised IAI)

In this approach, the ISIs are completely distributed and each ISI collaborates with the others for sharing data through a direct communication. The analytics services provided by the IAI use the data stored on the local ISIs.

3.4. Fully distributed: On-Premises ISI and IAI

In a fully distributed architecture, both the ISI and the IAI are deployed on-premises (see below).

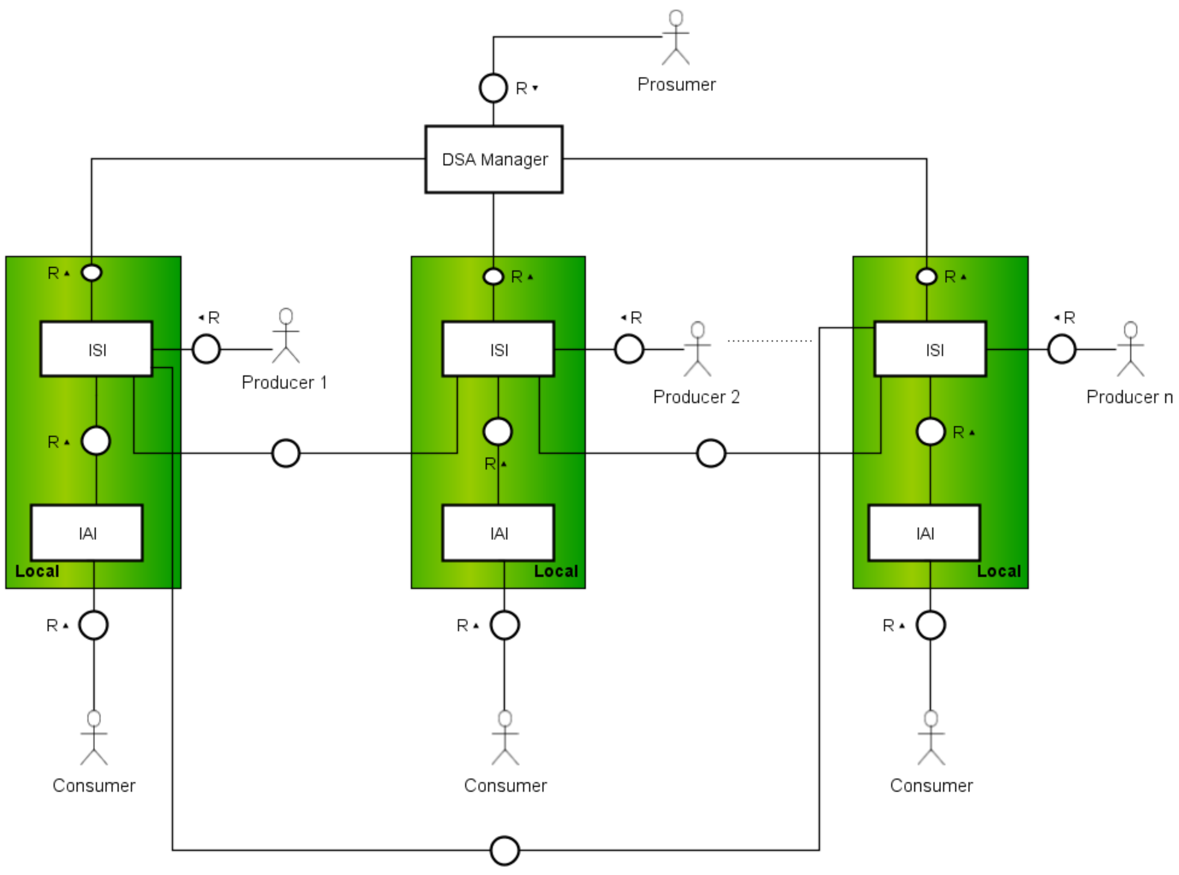


Figure 7: Fully distributed deployment model (On-Premises ISI and IAI)

The ISIs share data between themselves according to the established DSAs. The IAIs elaborate the analytics services locally by using the shared data stored on the corresponding ISIs. Differently from the others deployment models where the IAI is always centralised, in this case the Virtual Data Lake used by the Legacy Analytics Engine is locally instantiated on each IAI.

3.5. Instantiation of the Architecture in the Pilots

In this section, we provide an overview of the deployment models that have been chosen by each C3ISP Pilot in order to satisfy their stakeholders' requirements as well as to implement their use cases. Table 1 summarises the model chosen by each Pilot.

Table 1: Deployment models in the Pilots

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

The following subsections are dedicated to describing these choices and how the C3ISP architecture fulfils the Pilot requirements. More details about the deployment model adopted by each Pilot as well as the integration of Pilot-specific components in the C3ISP Framework can be found in D6.2 and in each pilot's deliverables (D2.2, D3.2, D4.2 and D5.2).

3.5.1. ISP Pilot

The ISP Pilot is concerned with the sharing of cyber threat information (CTI) among the Italian ISPs and the *Registro.it* (body responsible for managing Italy's top-level domain names), in order to mitigate possible attacks. Each ISP will use services, such as the list of malicious IPs, provided by the *Registro.it* to perform security checks, and will produce security reports that will later be shared with other ISPs via the C3ISP Framework. It is in each ISP's interest that they can define their own policies and determine in which form the data (i.e. security reports or logs) can be shared, i.e. whether the data needs to be encrypted or anonymised beforehand in order to protect sensitive information related to the ISP or its customers. An ISP can be seen as an isolated entity with enough resources to deploy some of the C3ISP components locally. Therefore, the *hybrid deployment model* is chosen in the ISP Pilot where each ISP will host a local (on-premises) ISI and remotely communicate with the centralised ISI and IAI subsystems hosted by a C3ISP service provider.

3.5.2. CERT Pilot

The CERT Pilot is concerned with fostering cyber threat information sharing between the Italian CERT and other C3ISP stakeholders, in particular ISPs and Enterprises, with the aim of preventing or timely reacting against security attacks. The CERT Pilot will adopt the *hybrid deployment model* because its Prosumers, e.g. ISPs or large enterprises, may decide to sanitise and cleanse their data prior to sharing it with the CERT, and thus need to deploy a local ISI. Other C3ISP subsystems (remote ISI and IAI) will be centrally hosted at the CERT's premises to perform tasks such as collaborative data analysis, protected data storage and DSA policy enforcement.

3.5.3. Enterprise Pilot

The Enterprise Pilot is concerned with providing a multi-tenanted managed security analytics platform that would allow controlled sharing or pooling of cyber security data belonging to different enterprise customers, without disclosing customer sensitive information. Since all the data, which has previously been collected from remote enterprise premises, is stored centrally at the Managed Security Service provider's premises (i.e. on a multi-tenanted data lake), the *fully centralised model* is chosen for the Pilot deployment. All relevant C3ISP components for data sharing and analytics will be hosted within the Managed Security Service Providers (MSSP)'s own platform and trusted domain (i.e. no external C3ISP provider). It is anticipated that each enterprise customer (or MSSP analyst working on behalf of the customer) will be able to define their own DSA policies using a DSA editor provided via the MSSP platform (i.e. customer portal).

3.5.4. SME Pilot

The SME Pilot is concerned with providing a managed security service in the cloud environment such as firewalls, intrusion detection/prevention systems, or anti-malware analysis, to the SMEs and the collection and sharing of SME cyber security data with the C3ISP service without disclosing privacy sensitive information. Participation of each SME in the C3ISP eco-system needs to be done seamlessly with as little effort as possible, which means that most of the required management and operational processes should be offloaded in order to minimise the utilisation of SME resources (i.e. software and hardware). Therefore, the SME

Pilot envisages the use of a Pilot-specific application service, called C3ISP Gateway, which should act as the middleware between the SMEs and C3ISP subsystems. The tasks for collecting, processing and sharing the CTI are delegated to the C3ISP Gateway. While each SME is still responsible for defining its own DSA policies, the policy enforcement should be performed by the C3ISP. However, the SME Pilot has also identified requirements similar to ISP and CERT Pilots where an SME wants to be able to sanitise and cleanse their data first before sending it to the C3ISP platform. For the above-mentioned reasons the SME Pilot will adopt the *hybrid deployment model* where local ISI may optionally be deployed in addition to the remote ISI and IAI (hosted by a C3ISP provider).

4. Subsystem: 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.

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

- DSA Adapter: to enforce the rules written in the DSAs;
- Format Adapter: to accommodate for different necessities of data format and conversion;
- Data Protected Object Storage: 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.

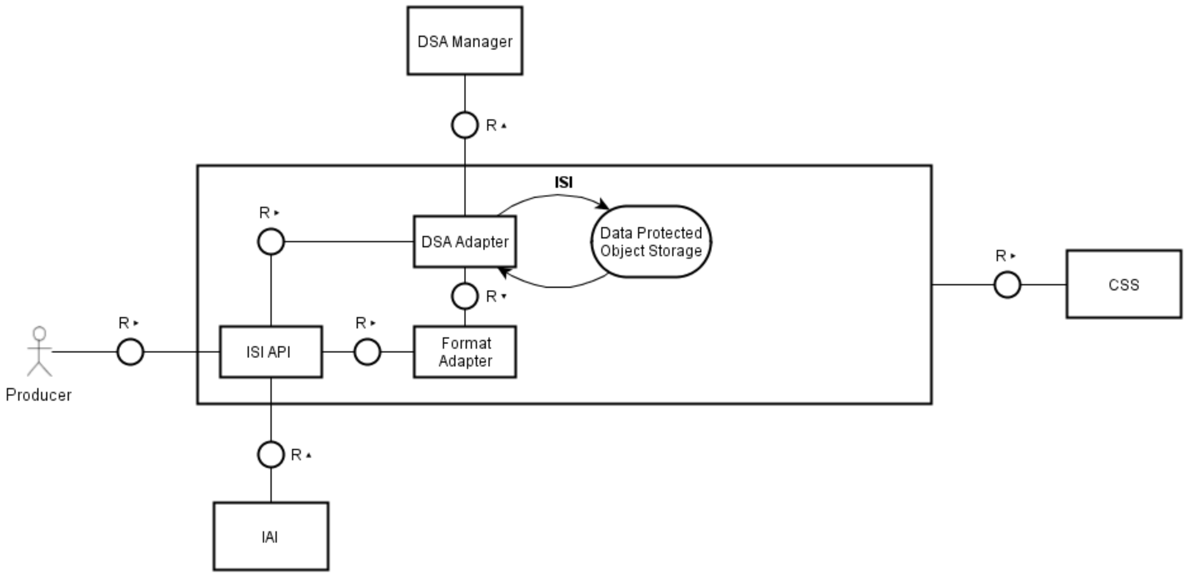


Figure 8: Information Sharing Infrastructure

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 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.

We adopt here the sticky policy approach [67], to pair the Prosumer's data with the DSA policies: this means that the policies are firmly associated[3] to a piece of data and the association is not breakable. Whether the datum is processed or transmitted, the sticky policy must be

[3] Whether it is the content of the policies themselves, references to policies which can be downloaded from a web site or a DSA id to be retrieved from the DSA Manager is dictated by performance rather than security reasons and can be considered an implementation decision.

processed and never be dissociated. We use the term *C3ISP Data Bundle* to refer to the CTI data and its paired DSA policies.

More precisely, the C3ISP Data Bundle (or bundle, for short) is an encrypted *Data Protected Object (DPO)* container to preserve confidentiality and integrity. The bundles are managed by the *DSA Adapter* component (with its *Bundle Manager* module), which is able to (i) pack/unpack the CTI data in the container structure and (ii) enforce the DSA policies specified. The next figure shows the C3ISP Data Bundle structure:

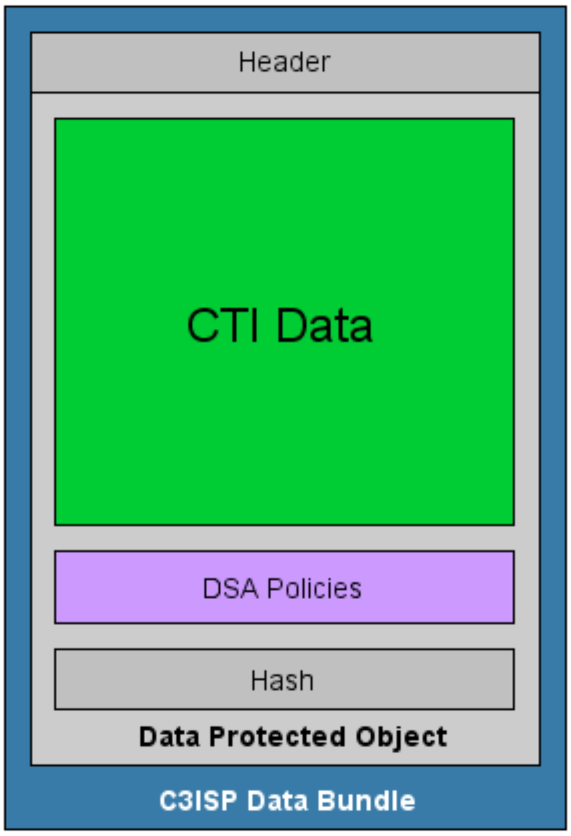


Figure 9: C3ISP Data Bundle

The bundle has an *Header* to provide identification and other bundle metadata (e.g., bundle format, sizing, etc.). The Data Protected Object is encrypted and contains the *CTI Data*, the *DSA Policies* and a signed *Hash* code to assure that the data has not been tampered with and that it is paired with the correct DSA policies. It is worth noticing that the encryption functionalities used to create the bundle are provided by the *Key & Encryption Manager* component (part of the CSS subsystem, see 7.2).

4.1. 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). In particular, since the DSA consists of a *Usage Control policy* (an enhancement to the traditional access control mechanisms, see Sect. 4.1.1), the DSA Adapter retrieves the attributes required for the evaluation from the other components of the architecture, evaluates the authorizations and conditions to decide whether the access can be performed or not, and performs the resulting obligations[4], which can even change the data itself before being released to the requestor.

Figure 10 shows the modules the DSA Adapter, and the following of this section provides a brief description of each module.

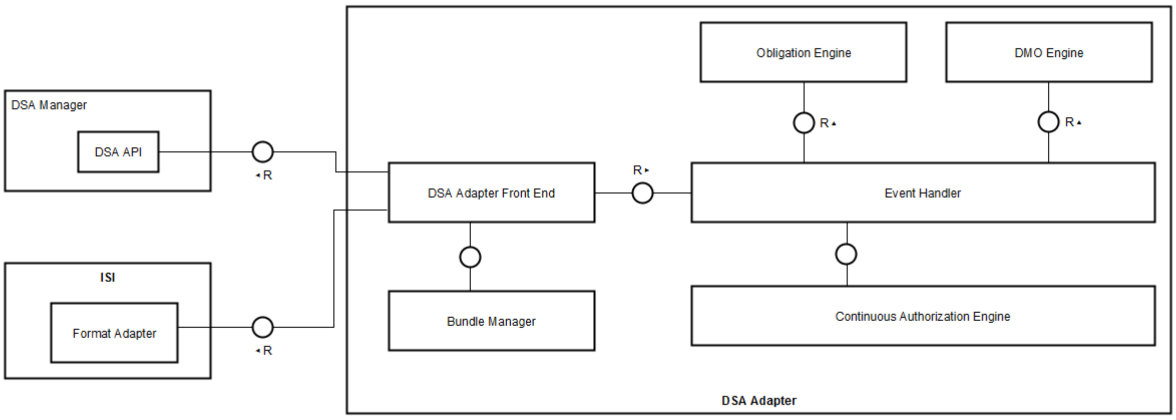


Figure 10: Components of the DSA Adapter

The **DSA Adapter Front End** is in charge of receiving incoming requests (from the ISI API) and to return the related answer to the requestor. The DSA Adapter Front End delivers each request it receives to the **Event Handler**, which is in charge of collecting and distributing the messages, also called events, resulting from the phases of the decision process (i.e., the partial results) among the other modules of the DSA Adapter. It allows other modules to register for events, and it notifies them when such events occur. On the other hand, it accepts event notifications from the other modules of DSA Adapter.

The DSA Adapter has three main sets of modules which implement its functionalities:

- Continuous Authorization Engine;
- Obligation Engine;
- Data Manipulation Operation (DMO) Engine.

The next subsections illustrates these modules.

4.1.1. Continuous Authorization Engine

The **Continuous Authorization Engine** (shown in Figure 11**)** is the engine which supports traditional access control (i.e., the authorization process performed at request time) and continuous access control (an enhanced feature introduced by the UCON model [64]). The traditional access control phase (called *preAuthorization* in UCON) enforces the security policy when the access request is received, in order to check whether the subject who requests the access actually holds the right to perform the action on the object. The continuous authorization phase (called *onAuthorization* in UCON), instead, checks that the right to perform the action continuously holds during the execution of the action itself, in order to take a countermeasure (such as interrupting or suspending the execution of the action) as soon as this right expires.

[4] Obligations are rules defining which Prosumers are obliged to perform which actions on which data, under a set of environmental conditions (see D 8.1 Sect. 3.1.1).

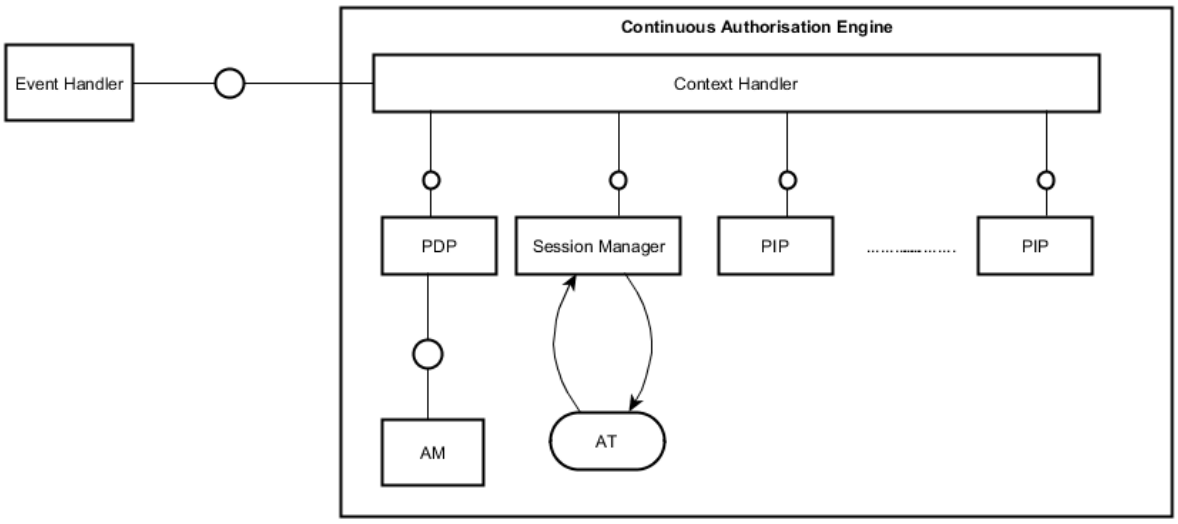


Figure 11: Continuous Authorisation Engine

The modules of the Continuous Authorization Engine are the following:

- **Context Handler** (CH) is the entry point of the Continuous Authorization Engine and it manages the protocol for communicating with the Event Handler. This protocol which regulates the interactions between the Event Handler and the CH is defined by a subset of the usage control actions: *tryaccess, permitaccess, denyaccess, revokeaccess*, and *endaccess*. The CH also coordinates the internal modules of the Continuous Authorization Engine for the execution of the policy evaluation process, as described in the remainder of this section;
- **Session Manager** (SM) is the components responsible for keeping track of the ongoing usage sessions, i.e., of the access that are currently in progress, and it exploits an **Access Table** (AT) to store the meta-data regarding these sessions. It is the key component of the continuous authorization phase, and it represents an extension with respect to the XACML reference architecture [64];
- **Policy Decision Point** (PDP) is the component which evaluates security policies and produces the access decision. In C3ISP Framework the PDP evaluates standard XACML policies because the usage control specific features are managed by the CH and by the SM;
- **Attribute Managers** (AMs) are modules which manage attributes, allowing to retrieve and to update their current values for running the policy evaluation process. AMs could be local, i.e., they run on the same machine as the Continuous Authorization component or remote, i.e., they could run on external servers that could be even located in other domains run by third-parties;
- **Policy Information Points** (PIPs) are interfaces for interacting with Attribute Managers in order to perform the following 3 main operations on attributes: *retrieve, subscribe/unsubscribe* and *update*. In general, the attributes required for the evaluation of a usage control policy are managed by distinct Attribute Managers, which require different protocols for interacting with them, and which provide different functionalities. For instance, some attributes are retrieved from other C3ISP components, other attributes are directly managed by the Continuous Authorization Engine component, while other attributes are managed by third organizations (e.g., an LDAP system). Hence, PIPs mimic a plug-in architecture to let the Usage Control

service be as flexible as possible in interacting with distinct and different Attribute Managers. In particular, the proposed architecture includes a set (chain) of PIPs which provide the same interface to the CH (*retrieve, subscribe/unsubscribe* and *update*), while each PIP implements the specific protocol to interact with a given Attribute Manager and the specific algorithm to perform the requested operation and to provide the required information. For instance, if the Attribute Manager of attribute A does not support subscription, the PIP paired with A should implement the subscription mechanism in order to provide the *subscribe* interface. This PIP could invoke periodically the Attribute Manager to retrieve the updated value and compare it with the previously collected value. If the new value is different from the previous one, the PIP notifies the component which performed the subscription (i.e., the CH). The time interval between two consecutive queries to the Attribute Manager is a configuration parameter and is set according to the attribute to be monitored. More complex techniques could be implemented. For instance, the previous PIP could exploit risk based techniques to decide how much time it should wait before retrieving the next attribute value from the Attribute Manager. The *retrieve* interface could be implemented by simply forwarding the request to the Attribute Manager, or by exploiting more complex techniques as well. For instance, the PIP implementation could return the attribute value retrieved in a previous interaction instead of retrieving a fresh value form the Attribute Manager if the risk that this value is changed is low. Moreover, the PIP could also be configured to act as a local cache to reduce the operation execution time when it is aware that the attribute value will not change for a while (e.g., when the PIP itself requested to the related Attribute Manager to lock the attribute).

4.1.2. 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.

The structure of obligations can be described as follows:

- Usage Control Obligation = do *Action* when *Trigger*

Where *Trigger* is defined by:

- *Trigger* = *Event* AND *Condition*

Therefore, an obligation results in the execution of a particular action, when a specific event occurs but only if a condition is verified.

Usage Control Obligations, therefore, may be defined by specifying the desired combination of actions and triggers. An initial set of triggers and action is currently supported by the actual implementation of the Obligation Engine, and through interactions with the Pilots' owners, this set may be extended and adapted as needed.

The following list of triggers is currently supported by the Obligation Engine. They are divided in two sets: general and time-based triggers.

General triggers:

- TriggerDataAccessForPurpose: Event-based trigger that occurs each time a specific data associated to the obligation is accessed for one of the specified purposes;
- TriggerDataDeleted: Event-based trigger that occurs when the piece of data associated to the obligation is deleted;

- TriggerDataSent: Event-based trigger that occurs when the data associated to the obligation is copied.

Time-based triggers:

- TriggerAtTime: Time-based trigger that occurs only once between start and start+maxDelay parameters;
- TriggerPeriodic: Time-based trigger that occurs every arbitrary interval of time, between start+maxDelay and end parameters.

The following list represents the supported actions at this stage:

- ActionDeleteData: This action deletes a specific piece of information;
- ActionNotifyProsumer: This action notifies the Prosumer when triggered;
- ActionLog: This action logs an event.

Internally, the Obligation Engine consists of a number of modules, as depicted in Figure 12:

- The **Obligation Handler**, in charge of processing (and persisting) the obligation definitions coming from the policies;
- The **Trigger Engine**, which supports multiple types of triggers by implementing their specific business logic;
- The **Action Engine**, that, similarly to the trigger engine, is responsible for materialising the actions in obligations;
- The **(Obligation) Event Handler,** that interacts with the (DSA Adapter) Event Handler to filter and process the events relevant to the Obligation Engine. In the following diagram, the Obligation Event Handler is indicated as Event Handler for convenience.

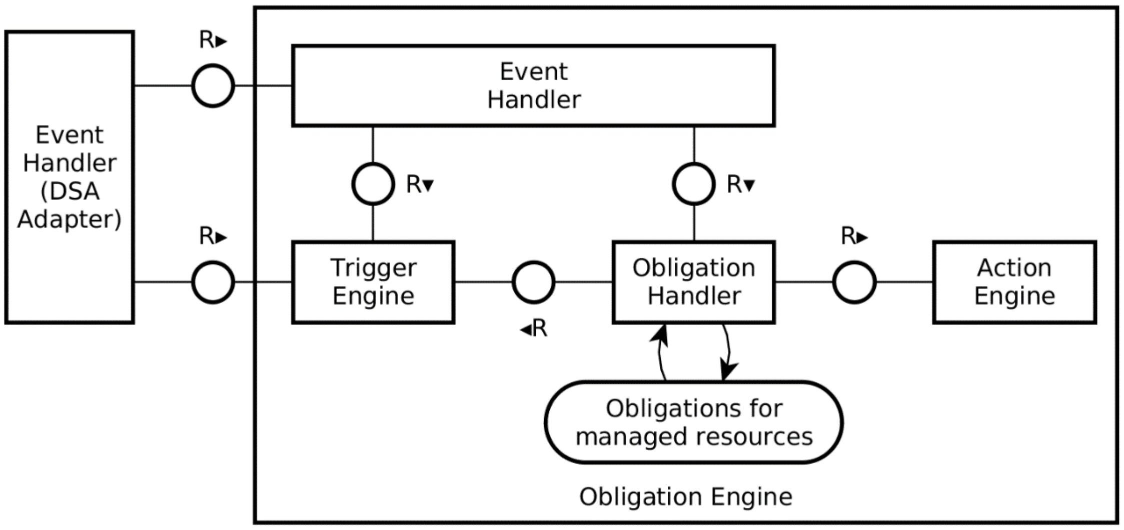


Figure 12: Obligation Engine Block Diagram

What follows is a description of the more relevant methods of the Obligation Engine modules, in order to provide a general overview of the internal dynamics.

First, the Obligation Handler has two operations, to add and to remove obligations. The Obligation Handler persists the obligations in a database, for efficiency. If an obligation includes a *time-based trigger*, the Obligation Handler calls the TimeBasedTriggerHandler.

The TimeBasedTriggerHandler keeps a list of TimeBasedTriggers. When called, the TimeBasedTriggerHandler instantiates a TimeBasedTrigger.

In order to implement their behaviour, all TimeBasedTriggers use a Java Timer[5]. When a TimeBasedTrigger ticks, it fires the corresponding event by calling the Event Handler.

The Event Handler offers an interface to trigger any event. When an event is triggered for a specific managed element, the Event Handler retrieves from the database the correct Obligation, and then calls the Action Handler to perform the associated action. Moreover, the Event Handler interacts with the Event Manager in order to route and pre-process any event of interest to the Obligation Engine.

The Action Engine offers an interface which determines the correct action to perform according to the specified Obligation.

4.1.3. 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.

The architecture of the DMO Engine is plugin-based, in order to support an arbitrary number of manipulation operations, and it is depicted in the following Figure 13:

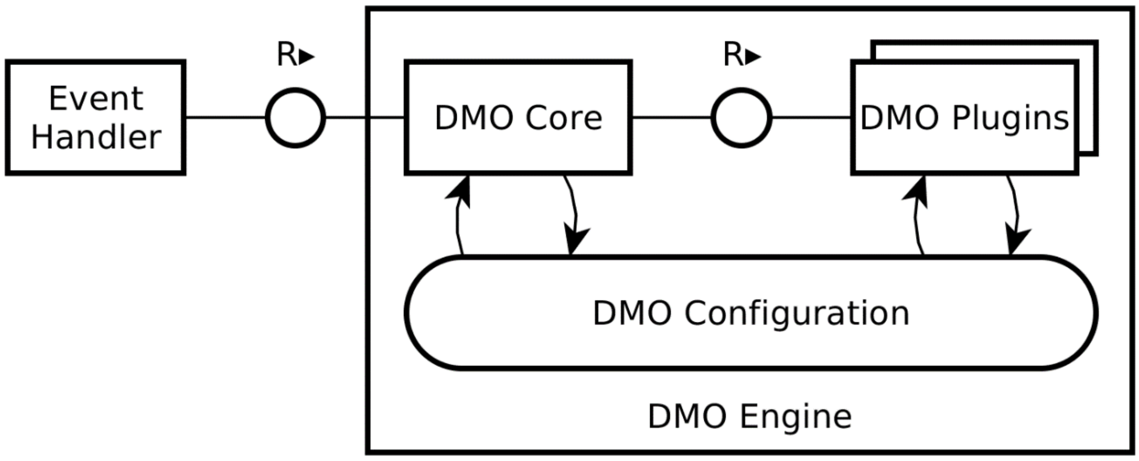


Figure 13: DMO Engine Block Diagram

- The DMO Engine connects to the DSA Adapter **Event Handler** in order to collaborate with the other components, e.g. with the Obligation Engine. The Event Handler mediates requests to the DMO Engine about mandated operations required by a DSA;

[5] Java Timer Class, http://docs.oracle.com/javase/8/docs/api/java/util/Timer.html

- The **DMO Core** is the component that interacts with the Event Handler on one hand, and invokes the correct data manipulation operation component;
- The **DMO Plugins** realise the actual manipulation, once invoked by the DMO Core. Plugins may be also external services, invoked as SaaS;
- The **DMO Configuration** stores the configuration of the DMO Engine; for example, it keeps track of the available DMO Plugins.

As mentioned, the DMO Engine will implement a set of operations which will cover the requirements of the C3ISP Pilots. In particular, we foresee at least the following list of DMOs:

- Anonymisation based on:
  - Simple attribute suppression/replacement;
  - Differential privacy techniques (e.g. to achieve geo-indistinguishability, see Appendix 1);
- Symmetric homomorphic-friendly encryption, used to pre-process data for applying homomorphic computations (see the *transcryption* process is 7.2.4.2).

4.1.4. Bundle Manager

The **Bundle Manager** is the module that handles the *C3ISP Data Bundle*, i.e., the container of the data protected object (see the bundle structure described earlier in 4). It is used for both packing and unpacking operations. In the packing phase, the Bundler Manager is used for creating a bundle by (i) selecting a DSA and then (ii) pairing the DSA with the provided CTI data. In the unpacking phase, instead, it is used to (i) extract the paired DSA from the bundle, which will be sent to the DSA Event Handler for policy evaluation, and (ii) to retrieve the CTI data (if allowed by the policies evaluation outcome).

As we described earlier, the C3ISP Data Bundle is planned to be a cryptographic container. For this reason, during these packing/unpacking operations, the Bundle Manager interacts with the Key and Encryption Manager to encrypt and sign the bundle at creation time, or decrypt and verify it when reading or accessing. The simplest *encryption schema* will be to use a symmetric encryption with a predefined key (per Pilot, since the C3ISP Framework is multi-tenant and should be capable to support more than one Pilot over the same deployment). Other encryption schemas could be available, e.g. to support the different trust boundaries of the Hybrid deployment model (see 3.2) where we have a local ISI and a remote ISI, we could have different encryption keys. Also, it could consider the usage of public key cryptography in other scenarios.

Another important capability of this module is the DSA selection at creation time. In fact when a Prosumer provides a CTI data to the C3ISP Framework, s/he expects that it will be protected by the policies he or she crafted in the DSA. There could be two different ways to address this need. The first is the *discretionary approach*, in which the Prosumer provides the DSA to be paired[6] when submitting the CTI data. The second is the *mandatory approach*, in which the Bundle Manager evaluates some logic to select the appropriate DSA directly from the DSA Manager (e.g. by considering some Prosumers-provided metadata like its username or the type of CTI data; by inspecting the content of the CTI data to infer the right DSA to use; etc.).

[6] This could be either the full DSA policies document or a DSA identifier/URI that will be resolved by the Bundle Manager at packing time.

Since both the CTI data and the DSA policies are regular system files, the implementation can opt to realise them, for example, in some sort of (signed) file archive format. As it will appear clear from section 6, the DSA policies stored into the bundle are those that have been mapped to the low level enforceable language (e.g. XACML) via the DSA Mapper component.

The Bundle Manager interacts with the Data Protection Object Storage (see 4.3) to store and retrieve the bundles.

The next picture illustrates the modules of the Bundle Manager:

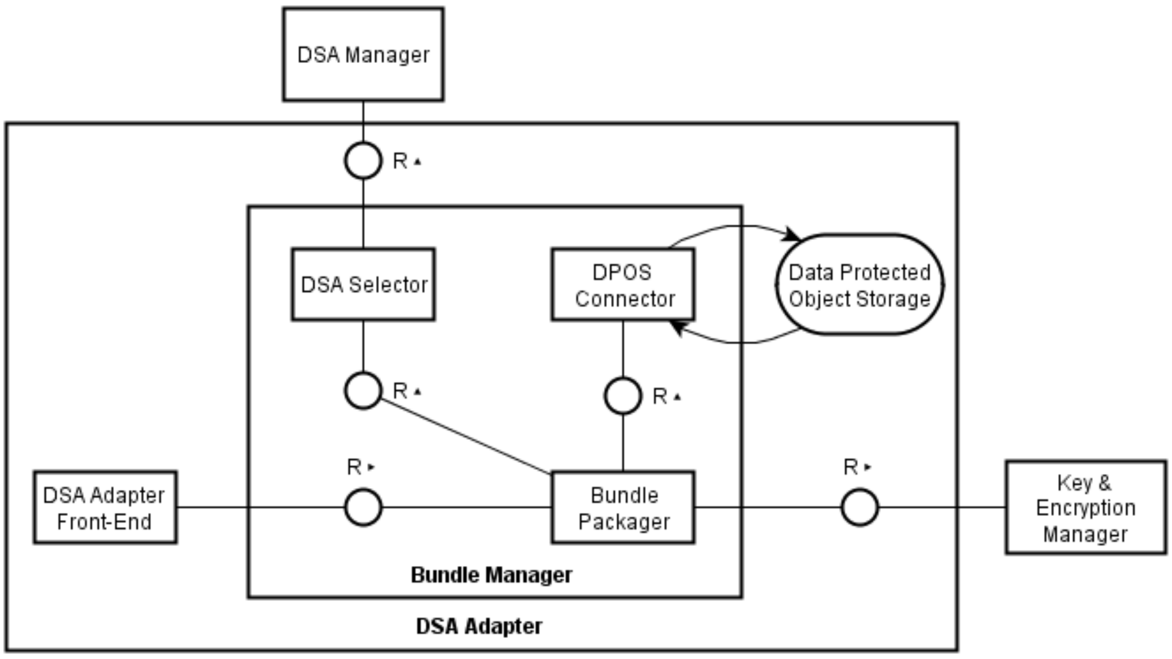


Figure 14 Bundle Manager in the DSA Adapter

The **Bundle Packager** is responsible for the packing/unpacking operations and interacts for this with the Key and Encryption Manager. The **DSA Selector** finds out the DSA to fetch from the DSA Manager that will be paired by the Bundle Packager. The **DPOS Connector** allows the Bundle Manager to connect to the specific technology used by the Data Protected Object Storage for storing and retrieving the protected objects (bundles).

4.2. Format Adapter

The C3ISP Framework is supposed to process data coming from different sources (Prosumers), coming thus in different formats, carrying different metadata and not necessarily respecting any standard for representation. In fact, we have already envisioned, also from the Pilot analysis, that data to be analysed will include CTI data (e.g., network logs, database entries, antivirus reports and emails). The **Format Adapter** is the component of the C3ISP Framework which adapts the format of data to a standard format to be easily processed by the various C3ISP components.

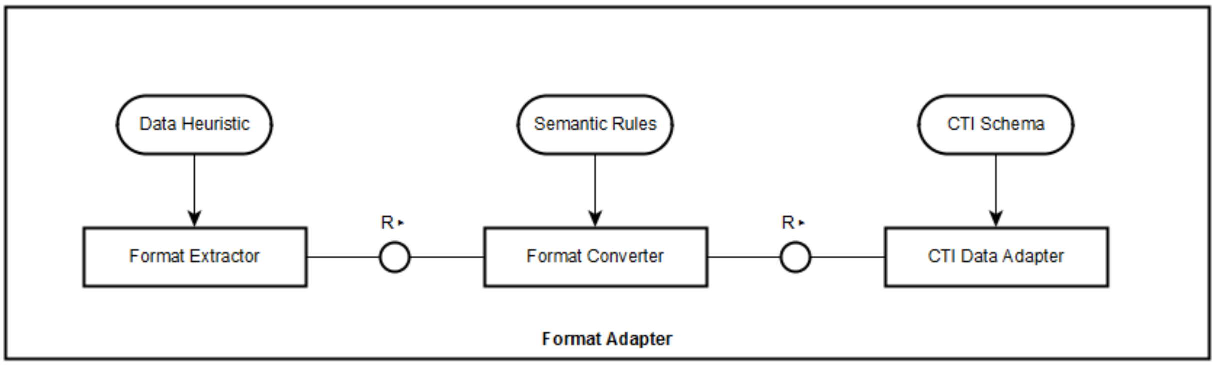


Figure 15: Format Adapter internal structure

In particular, C3ISP is designed to operate with structured Cyber Threat Information (CTI) represented in standard formats such as STIX, STIX 2.0 [2] or MISP[7], which can be easily parsed, imported as objects, have a powerful and expressive semantic, and give a user-friendly representation of the threat and related information. Figure 15 describes the internal structure of this module, representing the three logical operations that are performed to get structured CTI, starting from raw data. In particular, the *Format Extractor* is the component that includes the set of rules (heuristics) to automatically recognize the format of incoming raw data, in case the format is not already given by the data provider. The *Format Converter* gets as input the parsed data of the format extractor and converts it into a format which simplifies the data use for analysis, according to a set of semantic rules specified for the data format recognised by the Format Extractor. The converted data are then provided as input for the *CTI Data Adapter*, which is a configurable component whose task is to put the data in a structured standard format for CTI representation, such as STIX, STIX 2.0, MISP and other formats. The format selection is performed at set up time by the service administrator, and is unique for that specific instance of the C3ISP service. The standard CTI representation improves readability, also thanks to the presence of tools to visualize the CTI content in a user-friendly way. Moreover, this standard representation helps in defining a semantic for the represented pieces of information and their relation. CTIs are then stored as data protected object, ready to be used for analysis. Not all available algorithms for data analysis will be able to process structured CTIs, hence the CTI Data Adapter will be used again to remove the standardized structure, getting ready the data for analysis (as raw data).

4.3. *Data Protected Object Storage*

The **Data Protected Object Storage (DPOS)** persistently stores the CTI data provided by the Prosumers in the form of the C3ISP data bundle, i.e. the CTI data and the corresponding DSA. The DPOS could be implemented with an off-the-shelf object storage repository that allows storing complex documents (i.e. the C3ISP bundle) with additional metadata useful to query or retrieving them. Some examples of possible DPOS repositories include:

- *Standard filesystem*: it is a common filesystem that can be used as file storage; the advantage of a standard filesystem is that the read, write, share and file operations are fast and simple but the management of the metadata is not immediate (or if available, not easily portable between different implementations or filesystems).

[7] http://www.misp-project.org/

- *Amazon S3 Data Lake*: it is an object storage that can be used for storing and analysing Big Data, as Data Lake [52], hosting structured and unstructured data in a centralized repository. The advantage is the possibility to store data having multiple data types "as-is", without converting them in a predefined format. This is delivered only as a service by Amazon Web Services, Inc., so it cannot be installed on-premises.
- *MongoDB* [56]: it is a non-relational (No-SQL) document-oriented database [28]. It is free and open source. The documents are stored in a JSON-like format [53]; that simplifies the integration between different applications which share data as documents. Ad hoc queries, indexing, and real time aggregation features are provided for querying and analysing data.
- *Apache Hadoop Distributed File System (HDFS)* [54]: it is a distributed Java-based file system for storing large volumes of data across multiple machines. It is usually used to build Data Lake.
- *OpenStack™ Swift* [55]: it is the OpenStack™ module that implements an object storage service providing APIs to its managing. Swift is ideal for storing unstructured data that can grow without bound, so it is suitable for documents.

We plan to evaluate these solutions in particular considering that they should enable the runtime execution of analytics services provided by the C3ISP *Analytics Engine*.

4.4. 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 8). All the APIs are actions that are subject to the (DSA) policies associated to each CTI data object.

The list of the operations of the ISI API conceived so far are:

- **Create CTI**: used to submit a raw CTI data that will be paired with a DSA to create the C3ISP Data Bundle. In its simplest form, this operation can specify the DSA to be applied (e.g. via its DSA identifier, see 6.4);
- **Read CTI**: used to retrieve in raw format a previously stored CTI data from the ISI once the paired DSA policies have been met (e.g. access is permitted, a DMO for anonymisation has been applied, etc.);
- **Move CTI**: used to move a C3ISP Data Bundle from an ISI node to another (e.g. in the hybrid deployment model described in 3.2);
- **Delete CTI**: used to remove a CTI that have been previously stored (if DSA policies allow that action).
- **Prepare Data**: used for preparing data for performing analytics service; it implies that the CTI is retrieved (through the input CTI-Id) and formatted; the method returns a reference to the prepared data.

The basic signature of these API operations expects that the CTI data is provided as input at creation stage (Create CTI), while a CTI identifier (CTI-Id) is returned that can be used to refer to the stored CTI data in other operations (read, move, delete). At this stage, we do not plan an update operation, which can be realised as a delete followed by a create: we think this makes sense, since the CTI data is typically an object that is not to be modified after it has been created at the source (e.g. a log file when produced does not need to be altered).

The Create CTI is also used to submit to the C3ISP Framework the result of the analytics service run via the IAI subsystem. We specifically identify this variant because the DSA policies that the DSA Manager will need to pair to the CTI result data might be different from those of the CTI data used by the analytics service (see *Policies for Derived Data* concept in section 6).

5. Subsystem: 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 exploits the full capabilities of the C3ISP Framework, and so called legacy analytics service (i.e. already existent analytics), that can run on the shared data but have limitations.

The IAI is made up of the following components, described in details 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.

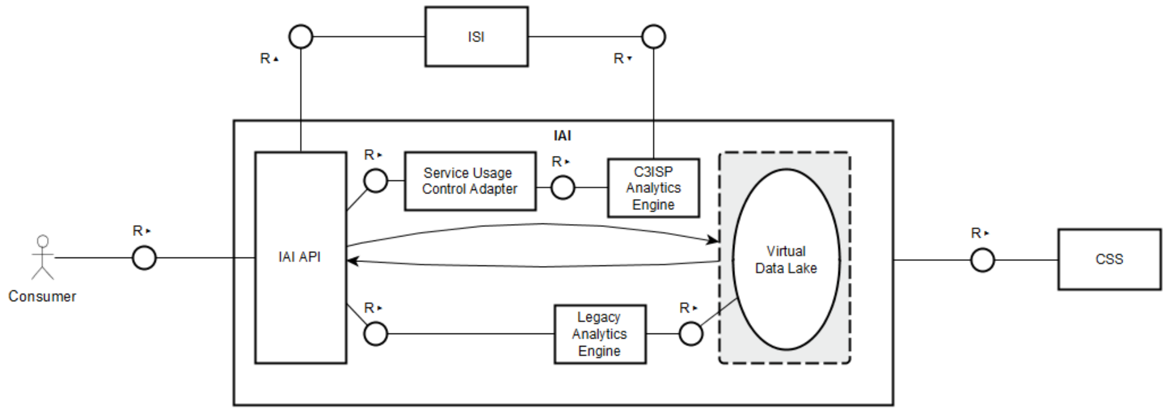


Figure 16: Information Analytics Infrastructure

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.

5.1. 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[8] and Scikit-Learn[9] libraries, and tools for Big Data analysis mainly derived from the Apache Hadoop[10] software suite, such as Spark[11], Flink[12] and Mahout[13], or internally implemented by project partners (e.g. homomorphic encryption, see 5.1.1 and 7.2.4.1), either as output of research activities in data analysis, or as a commercial product.

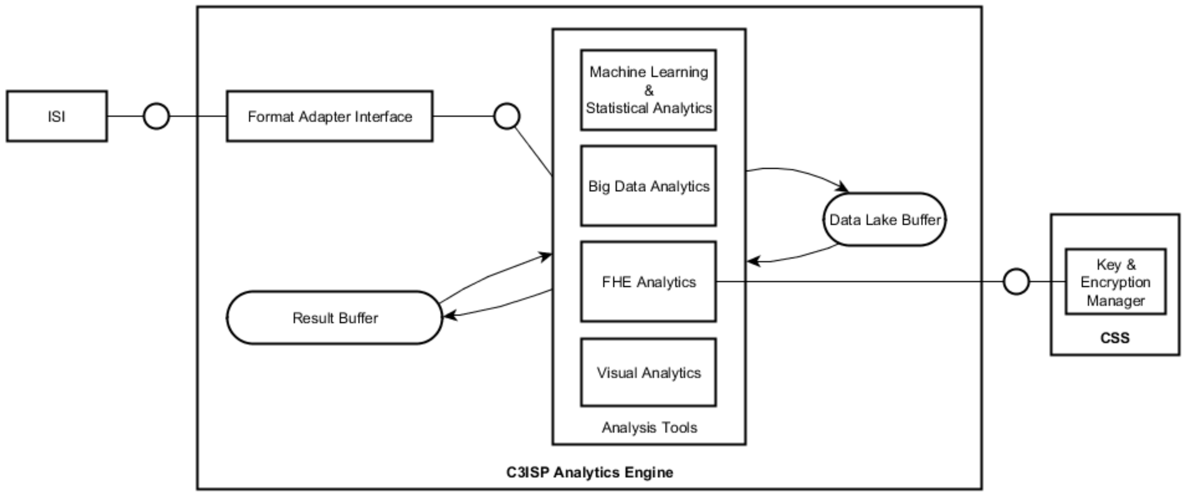


Figure 17: C3ISP Analytics Engine

The infrastructure depicted in Figure 17 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. 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.

The **Analysis Tools** are divided in the aforementioned sets:

- Machine Learning & Statistical Analytics;
- Big Data Analytics;

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

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

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

[11] https://spark.apache.org/

[12] https://flink.apache.org/

[13] http://mahout.apache.org/

- FHE Analytics (see 5.1.1);
- Visual Analytics (see 5.1.2).

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.

5.1.1. FHE Analytics

The **FHE Analytics** module is in charge of performing *homomorphic computation functions* (HE computation or FHE analytics – for short) on homomorphic encrypted data (more in this section). The homomorphic encryption process is handled by the Key & Encryption Manager component services, detailed in 7.2, so please refer there for all the details, in particular the FHE Analytics uses the *Homomorphic Encryption with Transcryption* process. The high-level process is following:

- A Prosumer submits a CTI data to the ISI;
- Before being persisted in the DPOS, a specific DMO[14] is executed on the CTI data (as prescribed by the DSA policies, which "knowns" that the data needs to be prepared for an HE computation) – the DMO is a specific symmetric encryption that is *homomorphic-friendly* (see for example Kreyvium in 7.2.4.2). This DMO retrieves the required cryptographic key (called *evaluation key*, a public data, see 7.2.4.2) from the Key and Encryption Manager;
- A Prosumer asks IAI for a FHE analytics service on the CTI data;
- The C3ISP Analytics Engine retrieves the CTI data via the Format Adapter Interface and passed it to the FHE Analytics;
- The FHE Analytics interacts[15] with the Key and Encryption Manager (FHE K&M Manager, in particular) to perform the *transcryption* process that transform the CTI data from the symmetric *homomorphic-friendly* encryption to homomorphic encrypted data;
- The FHE Analytics runs the HE computation over the homomorphic encrypted data producing an encrypted *result* (for this it needs the same *evaluation key* from the Key and Encryption Manager), which is finally sent to the ISI for being persisted as a new CTI data (via the Create CTI ISI API);
- Finally, a Prosumer retrieves the result (via the Read CTI ISI API). Since the result is encrypted, the Prosumer either needs to get the secret key from the Key and Encryption Manager, or s/he could receive the CTI data already in the clear (thanks to a variant of the Read CTI which decrypts on-the-fly on the C3ISP Framework-side). Both options will be evaluated.

[14] While FHE Analytics will be agnostic with respect to the C3ISP Deployment Model, we think that the common scenario will be to use this DMO for HE computation on hybrid models, where this DMO will be executed locally on the Prosumers premises to assure the maximum level of confidentiality.

[15] As it will be clear later, homomorphic encryption is a resource intensive process, in particular of disk space: for this reason, the FHE Analytics and the Key and Encryption Manager should be co-located in the C3ISP deployment model, to avoid huge data transfers.

The diagram in Figure 18 shows the FHE Analytics and the ISI (specifically via the ISI API) and the Key and Encryption Manager (specifically the FHE K&E Manager via the K&E Core).

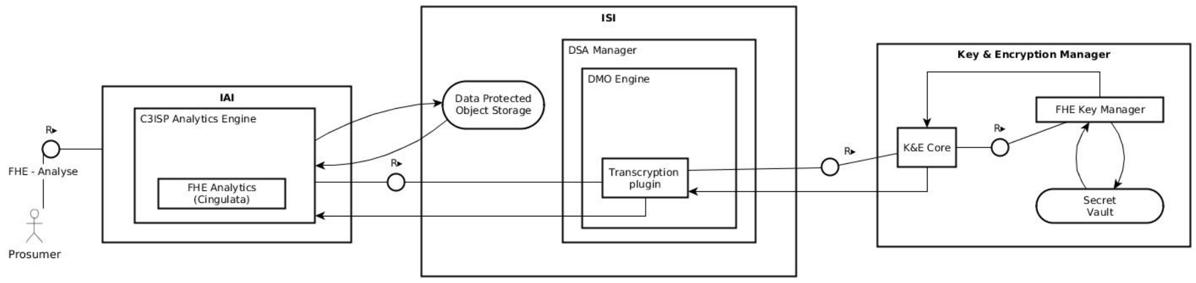


Figure 18: Analytics service with FHE Analytics

The benefit of homomorphic encryption is to permit analytics on encrypted data and to preserve data privacy at the same time. To achieve this, the encrypted data is not decrypted during the analytics operations. Moreover, the result of analytics with homomorphic encryption is still encrypted. In the C3ISP project and due to Pilots requirements, we propose FHE analytics services on two data types:

- IPv4 addresses;
- ASCII strings.

We will implement different **FHE analytics services** which will allow the following operations:

- Testing if two encrypted IPv4s are equal or not;
- Testing if an encrypted IPv4 belongs to a list of encrypted IPv4s (e.g. to check if the IP is in a list of malicious IPs);
- Computing the intersection of two lists of encrypted IPv4s;
- Computing the number of occurrences of an encrypted IPv4 in an encrypted list;
- Testing if an encrypted string belongs to a list of encrypted strings, where all strings have the same length;
- Testing if the *maxlen* first letters of an encrypted string belongs to a list of encrypted strings, where strings can have variable length and where *maxlen* is a parameter of integer type (e.g. to check if a hostname is in a list malicious hostname, if a username is in a list of sensitive accounts, etc.).

The analytics operations will be performed with **Cingulata** (originally Armadillo [11]), a tool from CEA. HE computation services on (homomorphic) ciphertexts, including the FHE analytics services proposed above, can all be decomposed on the elementary operations of homomorphic additions and homomorphic multiplications (can be performed with Cingulata tool) over the input bits of data. How this works is explained in Appendix 2: Homomorphic Computation.

5.1.2. Interactive 3D Visualisation

In addition, C3ISP provides a 3D visualisation pipeline that is able to display Internet connections based on IP address geolocations and thus point out the sources of potential attacks and the amount of traffic generated. Using snapshots of data from the Data Lake retrieved via e.g. a Hadoop connection, the visualisation platform generates geographical information systems (GIS) visualisation in 3D whereby the start and end point of connections denote the origin and destination of internet traffic retrieved by the system. Visualising such information in 3D over a world map provides better overview of large amounts of data since further any two points in geospace are, the larger the arc becomes. This offers a distinct advantage over plain 2D visualisation as many connections over time tend to clutter the view making such a system unusable, see Figure 19 for an example. Furthermore, using the online platform of 3D Repo, it is possible to gain access to such 3D visualisations directly in web browsers in 3D without the need to install any additional software. The same visualisation can also be displayed in Virtual Reality for simplified navigation and full immersion. Individual layers of data traffic can be switched on and off so that flows can be isolated and examined in more detail. Specific colour-coding further designates data traffic that fulfils certain criteria coming from the Analytics Engine, such as a common origin, matching connection signature, etc. All such data is then stored in a MongoDB [56] and pre-processed for fast web-based visualisation using the Unity3D [58] game engine.

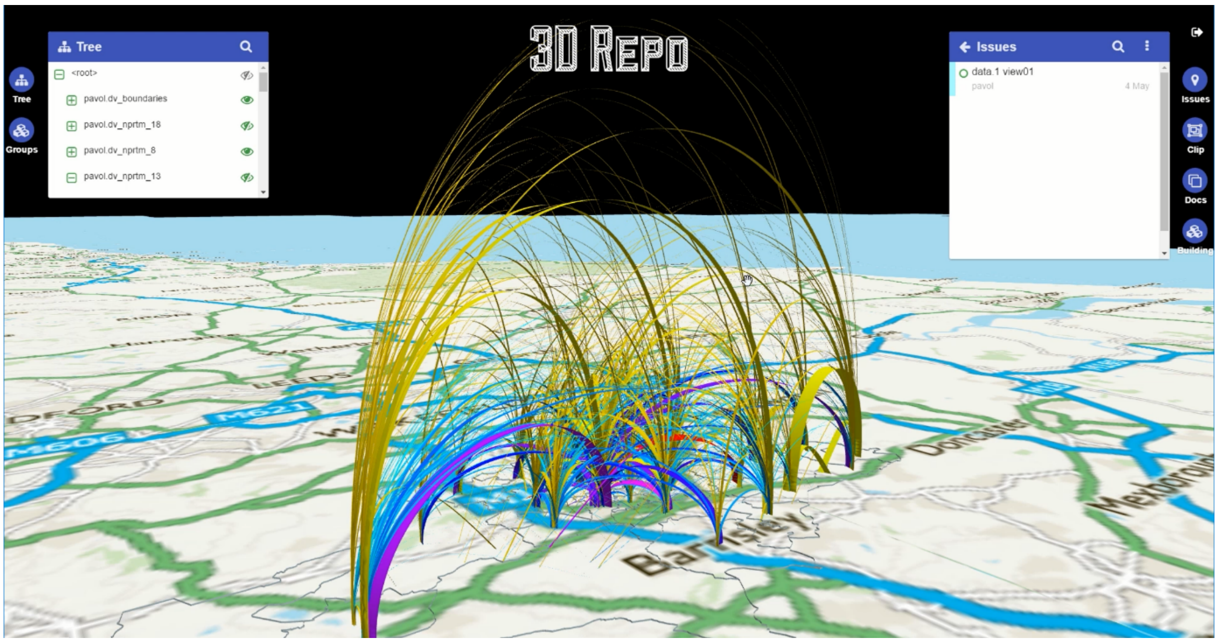


Figure 19: Mock-up data analytics visualisation in 3D

5.2. 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 5.5) from unauthorized accesses and usage. This adapter, similarly to the *DSA Adapter* previously described (Sect. 4.1), 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.

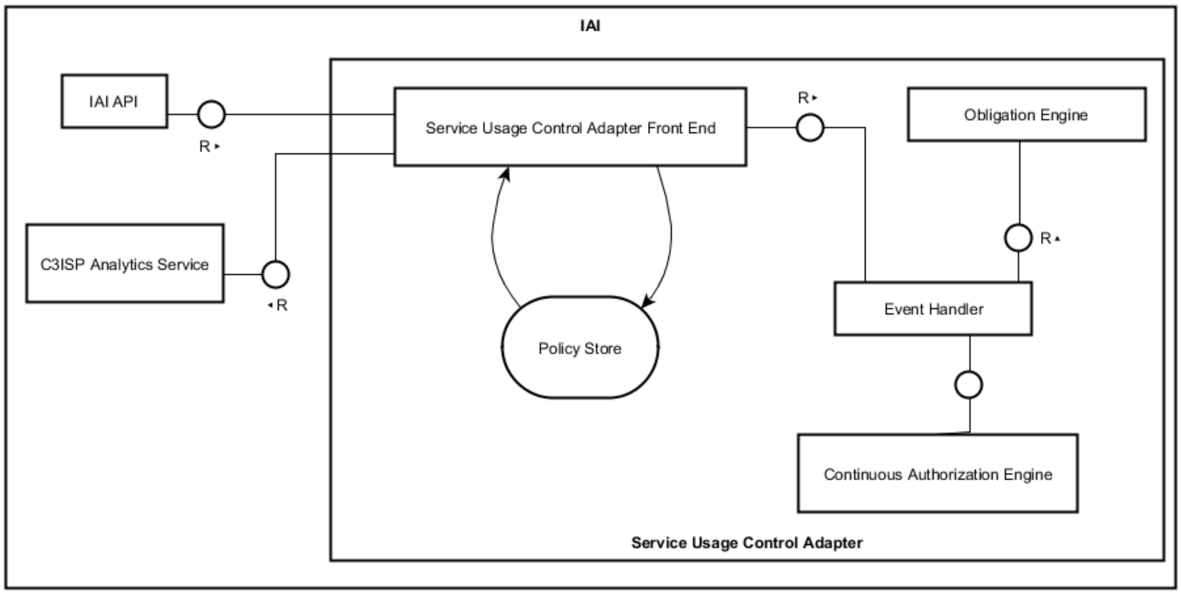

IAI
IAI API
R
Service Usage Control Adapter Front End
R
Obligation Engine
C3ISP Analytics Service
R
R
Policy Store
Event Handler
Continuous Authorization Engine
Service Usage Control Adapter

Figure 20 Service Usage Control Adapter

5.3. Interface to Legacy Analytics Engines

Legacy Analytics Engine such as the SATURN Visual Analytics tool (see D8.1 for details) is integrated as standalone web-application into the C3ISP reference architecture and accessible via user's web browser. The engine exposes interfaces to the following two C3ISP components:

- IAI API;
- Virtual Data Lake (VDL).

The interface to IAI API allows the invocation of legacy analytics service as well as provisioning of its result. It is assumed that a legacy analytics engine will provide its own graphical user interface, and thus its service consumption consists of sending an HTTP request to a specified location (URL) and receiving an HTTP response from that location (i.e. HTML page). Depending on the type of HTTP method used in the request, i.e. GET- or POST-method, additional parameters will be included in the HTTP request header or the message body. Legacy analytics engine may have its own authentication and authorisation component for restricting access to authorised users (consumers) as well as to data and features according to users' group and/or role. There may be an opportunity to integrate this component with C3ISP-own authentication framework in order to allow single sign-on to the legacy analytics engine. In this case, some user identity information and a valid session or authentication token will be provided as parameters when invoking the service via IAI API and another interface between legacy analytics engine and C3ISP authentication component (included in C3ISP *Identity Manager*, part of the CSS C3ISP subsystem) will be established to validate the corresponding token. If no single sign-on is implemented the consumer's service request would be redirected to the legacy analytics engine's own authentication page and the authentication/authorisation process will be carried out irrespective of C3ISP's own authentication framework; the authentication database of both systems (i.e. user IDs) need to be synchronised accordingly in order to ensure that each consumer can only access data available to them. The authentication may only be performed once when a consumer invokes the legacy service for the first time during a C3ISP session and may last until the consumer logs out from C3ISP system or after specified timeout.

The interface to Virtual Data Lake (VDL) allows the legacy analytics engine access the data that has been shared through the ISI and processed by other C3ISP components such as C3ISP analytics engine or data manipulation operation engine according to the DSA rules for particular consumer. Basically, the legacy analytics engine needs to define a data source connection to a VDL instance that was created upon consumer's request. Depending on the configuration options of the legacy analytics engine such connection setup may need to be permanent or pre-configured for each consumer or group/role, while at the same time the availability, type and amount of data accessible via the VDL instance may change each time a consumer submits a new analytics function request. The connection setup usually consists of the following parameters: data source type and name, IP address, port number, access credentials and data query information.

5.4. Virtual Data Lake

The **Virtual Data Lake (VDL)** is a dedicated service instance for storing CTI data in raw format to be consumed by a Legacy Analytics Engine. Since legacy analytics engines are not expected to support DSA enforcement on shared CTI data, any data accessible via the VDL instance must have already been prepared and pre-processed by appropriate C3ISP components (i.e. DSA Adapter and Format Adapter) according to the DSA rules and usage constraints (e.g. data sanitisation). This means that part of the data may be anonymised (thanks to DMOs specified in the DSA) or they are accessible only to particular groups or roles of consumers for limited period of time (thanks to a DSA policy rule). Hence the existence of a VDL instance as well as the contained data is still controlled by the DSA rules enforced by the ISI subsystem. Whenever the CTI data is processed and transferred to the VDL, the data on the DPOS should be flagged as *tainted*, because it has been used outside the control of the C3ISP Framework (and the taint flag could be used to defined specific DSA policies on such data, if needed).

The VDL will provide a standard SQL-like interface for querying the data to be used by the Legacy Analytics Engine (see 5.3). The following systems may be used for implementing the VDL:

- *Relational Database Management System (RDBMS)*: this comprises conventional database systems such as Oracle [74], MySQL [73], or PostgreSQL [68], which mainly employ SQL as their query language. The VDL can be instantiated as a table view to represent a subset of the sanitized data contained in the database which has been made available by C3ISP to specific requesting Prosumer;

- *Apache Hadoop Distributed File System (HDFS)*: this is a scalable Java-based file system that can be used to store huge amount of data on a cluster of servers. A VDL instance may be implemented as an HDFS folder which contains the sanitized data for specific Prosumers. The SQL-like interface to query the data (e.g. log files inside the HDFS folder) will be provided through off-the-shelf tools like Apache Hive [62] or Cloudera Impala [63].

The choice of which system to use mainly depends on the available resources (e.g. hardware, software, technical expertise) as well as on the amount of data to be stored. Given that the HDFS-based solution may require more effort to implement, it may suit better in a large-scale C3ISP deployment with high number of Prosumers interacting with the legacy analytics engine. We will take this into account when implementing the C3ISP (Pilot) testbed.

5.5. 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 CTI data (or multiple CTIs) shared and provided by the Prosumer(s). For this reason, the API includes:

- A method for invoking an analytics service (*runAnalyticsService*), which will receive as input the service analytics name (in the form of a unique identifier) and the CTI identifier (CTI-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 CTIs, i.e. two or more CTIs 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 CTI-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 CTI) 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 data in a dynamic way.

6. Subsystem: DSA Manager

As explained in the D7.1, C3ISP Framework uses Data Sharing Agreements (DSAs) for regulating information exchange across it, or rather data sharing policies encoded in the DSA regulate the communication between Prosumers.

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.

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: to persist the DSAs on a storage area;
- DSA API: to manage the DSAs from the other subsystems.

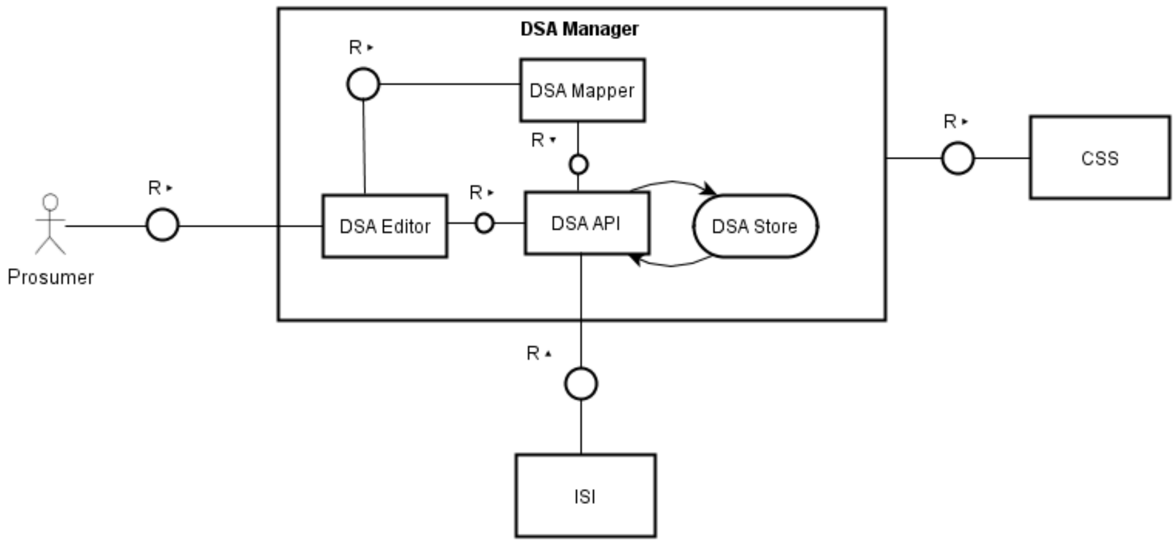


Figure 21: DSA Manager

The DSA Manager interacts with external clients:

- the Prosumer, which would use the services exposed by the DSA Editor;
- the Information Sharing Infrastructure (ISI) that needs to use the DSA API 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.

6.1. DSA Editor

The **DSA Editor** is responsible for orchestrating the communication between the other components of the DSA Manager. It is provided as a web application and supports the DSA life-cycle, from its inception and updates (through the DSA Editor), to its termination, as explained in D8.1, section 3.1.

The DSA Editor allows authoring DSAs and supports the Prosumers in the definition of data sharing policies. The DSA Editor provides a refinement process to create and manipulate the DSA: it allows the user to define an "abstract" and generic version of the DSA, called DSA Template, where only a reusable set of fields and rules are written. An "instance" of the DSA Template must be defined in order to obtain a DSA that can be used. DSA Template acts as a starting point for creating use case specific DSAs, in which all fields and rules are fully specified. More concretely, a DSA Template is like a generic contract that might be used by some parties to define the sharing rules in a specific context (e.g. a regulated marketplace) without starting from scratch: it is however evident that the parties could have specific necessities that are not part of the template and which shall be specifically written in the final DSA (i.e. the DSA instance).

Templates can build a catalogue of DSAs created for certain contexts and needs that can be used for defining a DSA without starting with a blank page.

The DSA is formalised in an XML (Extensible Mark-up Language) file format. The structure has the following main sections, which will be extended to accommodate the C3ISP Framework needs (e.g. to manage CTI data and the Prosumer network):

- The **metadata**: a unique identifier, a title, the data classification of the information protected by the DSA, the purpose of the agreement, the temporal validity of the DSA;
- The **policies** which express rules about authorizations, obligations and prohibitions; they are encoded in a Controlled Natural Language (CNL) [26], based on predefined dictionaries that are use case specific;
- Additional data for describing (in a free-text format) the content of the DSA or to add notes.

The DSA Editor supports the user in the definition of the policies, which represent the core of the DSA; the DSA Editor suggests the user only the possible terms and actions in an interactive and dynamic process during the definition of the policies. This feature leverages on a dictionary (also called vocabulary), based on an **ontology** defined in OWL (Ontology Web Language) [27], which describes the domain in which the DSA will be applied (it means that the dictionary could be Pilot-specific and the ontology shapes terms and actions of a specific context) and, in particular, the actions supported by the C3ISP Framework including **data sharing**, **data manipulation operations (DMOs)** and **analytics functionalities**. The user can define policies on data shared/provided by the Producers to the C3ISP Framework and also **policies on derived data**, that is on the CTI data resulting from running the analytics service on such shared data. We plan to define dictionaries for all the C3ISP Pilots.
The application supports different roles for creating DSA Templates and DSAs, and the layout of the graphical web interface changes according to a role-based authorisation mechanism where, for example, some fields cannot be changed when a DSA is instantiated from a template. The role-based model will be extended to support C3ISP security requirements (see D7.1, 2.2.1).

The following picture summarises the steps just described to support the DSA authoring process:

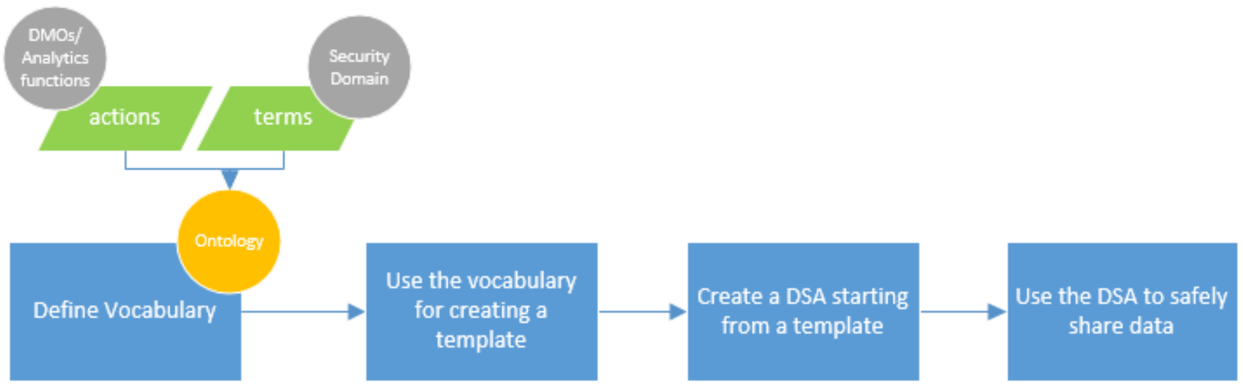

DMOs/ Analytics functions
Security Domain
actions
terms
Ontology
Define Vocabulary
Use the vocabulary for creating a template
Create a DSA starting from a template
Use the DSA to safely share data

Figure 22: DSA Editor Process

Once the policies are authored by the Prosumer and the DSA is finalised, the rules encoded in CNL (wrapped in the DSA xml file format cited above) are ready to be submitted to the DSA Mapper for translating the CNL statements in a low-level language suitable for enforcing them on the C3ISP Framework. This integration shall occur using web services technology (e.g. RESTful calls).

The DSA Editor also interacts with the DSA Store, the repository used to persist the DSAs (templates and instances).

6.2. 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). It is called by the DSA Editor to translate the DSA policy before the DSA will be stored in the DSA Store.

The DSA Mapper functionalities and how they work are described in D8.1, Section 3.2. The next figure shows its modules:

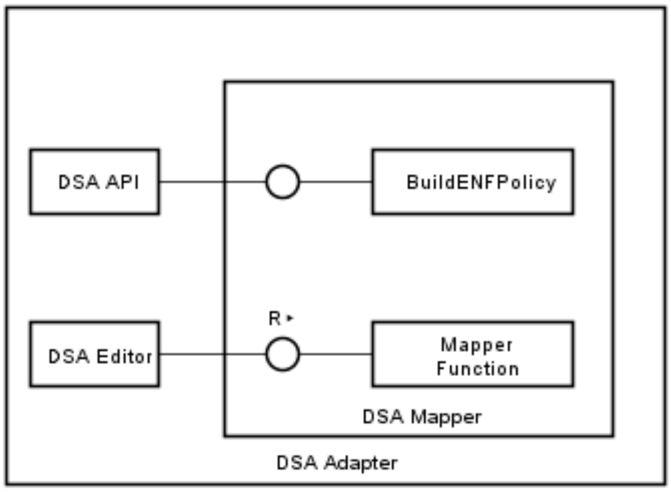


Figure 23: DSA Mapper

In a nutshell, the output of the mapper function is done in two-steps:

1. First, the DSA XML file is enhanced with the enforcement language specification of each rule composing the DSA (however this is not yet enforceable since it is not well formed and completed);
2. Second, the mapper builds a fully enforceable policy with all the constructs that were not added in step 1 (complete of both access control and usage control rules, as well as pre- and post-obligations) starting from a mapped DSA.

The DSA Mapper is internally called by the DSA Editor to translate a DSA policy from Controlled Natural Language to the enforcement language (XACML), before the DSA will be stored in the DSA Store.

Further, a "BuildENFPolicy" functionality (where "ENF" stand for "enforceable") is provided to extract the enforceable part from the DSA when needed (i.e. when required through the DSA API via the "Fetch enforceable policies from the DSA", see 6.4). This functionality is in charge of:

- Parsing the DSA XML file, retrieved from the DSA Store (via the DSA API), to:
    - Check the state of the DSA. In particular, if the DSA is not mapped yet, then the function halts. Otherwise the BuildENFPolicy starts the building procedure.
    - Isolate and extract the XACML rules from the DSA and save them separately.
- Building the enforceable policy skeleton, that represents the fix part of the enforceable policy. It is made of:
    - The header of the policy comprehensive of the syntactical details, including, for instance, the namespace, the combining algorithm, and so on;
    - Three specific tags for the description, the DSA identifier and the XACML policy Target.
    - A default deny rule to implement the default behaviour if no rule is matched.
- Including the rules derived from the DSA. Note that rules are inserted into the enforceable policy in a specific way: authorizations and prohibitions go first and then all the obligations. Obligation rules are built as rule that are PERMIT by default.

The DSA Mapper is provided as a Web Service of the DSA Manager subsystem. The current version implements APIs by exposing RESTful endpoints.

6.3. DSA Store

The **DSA Store** is a database where the DSAs are stored and consumed by the other components of the DSA Manager. Since it will contain documents (DSA are XML files), we are evaluating the use of a document-oriented database [28] (i.e. NoSQL database), in particular XML Databases which are optimised for storing, querying and managing XML documents.

The DSA Store will also be exposed, through the DSA API (see next), to the ISI subsystem to retrieve the appropriate DSA for the CTI data to be shared and protected at C3ISP Framework operation time.

6.4. DSA API

The **DSA API** is the external interface of the DSA Manager subsystem. It provides functions for managing the DSAs. The DSA API is used by the ISI for interacting with the DSA Manager when it needs to retrieve DSAs or by the DSA Editor when it needs to create or update DSAs.

At this stage, we foresee the following interfaces:

- **Create DSA**: used to persist a DSA in the DSA Store. The API shall return a unique DSA identifier;
- **Retrieve DSA**: used to retrieve a DSA, e.g. by its identifier, by its status, by other properties;
- **Retrieve the status of a DSA** (see DSA status in section 3.1 of D8.1);
- **Update DSA**: used to modify the content of an already existent DSA, by its identifier;
- **Delete DSA**: used to delete a DSA, by its identifier;
- **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 6.2) 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, etc. See section 8.1 and 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. Subsystem: CSS – Common Security Services

The C3ISP architecture requires some common services to be available to all the subsystems and their components in order to satisfy security requirements. These services provide a standard interface to the clients' components who want to use them. The following sections describe the subsystems suppling these services.

The CSS subsystem is made up of the following components, described in details 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.

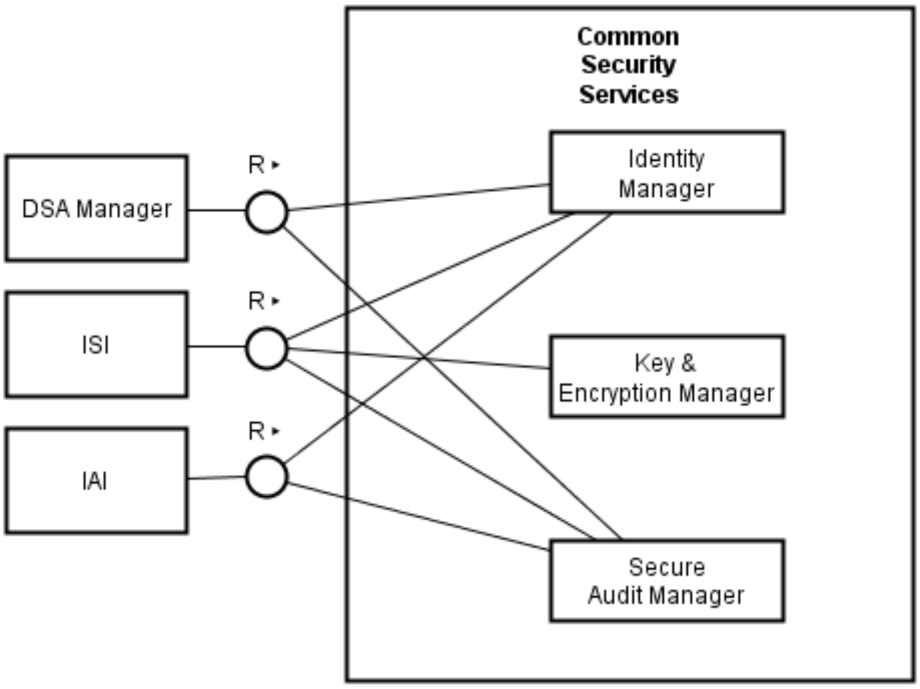


Figure 24: Common Security Services

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.

7.1. Identity Manager

The **Identity Manager** aims at identifying entities and storing authorization information within C3ISP. Entities are identified with the use of credentials that after verification allow or deny that entity to access the requested service. The basic-authentication system is the simplest technique to authenticate entities and it is composed of a username, which it is a public parameter, and a password (a secret), which must be known by the entity (client), which wants to be authenticated and another entity (server), which wants to verify the password and in case provide a service.

A more advanced authentication system is the Strong-Authentication, which adds to the basic authentication an additional parameter called One Time Password (OTP) [59]. An OTP is a secret that in freshly generated by an OTP-generator, and it is valid only once for a specific time-window. The OTP is generated by the client, which wants to be authenticated, and the server, which wants to verify the OTP generated. Both the client and the server use a common seed to generate the fresh OTP, like for instance the timestamp.

Other versions of Strong-Authentication use other channels to exchange the secret. For instance, it is possible to use the mobile telephone number of the client to receive a secret sent by the server that must be inserted during the authentication procedure. A variant of this protocol uses the email of the client to send the secret and then it verifies in the same way during the authentication phase.

Within C3ISP Framework, we envision a single Identity Manager which is able to verify credentials of identities that access different services also for different purposes. For instance, in the ISP Pilot a Prosumer must be authenticated to interact with the DSA Manager to create and manage DSAs. Also, interactions with the ISI and IAI may require an authentication step before proceeding with the services (see also API protection is section 2.1).

The *Identity Manager* will be also in charge of serving functionalities related to the authorisation phase. So, the *DSA Adapter* may query the *Identity Manager* to retrieve information for a specified entity, and use this information to enforce the action attempted by that entity.

There exist several standard for the authentication process, and we cite:

- **LDAP** [60] and LDAPS [70]: as defined by Microsoft, the "Lightweight Directory Access Protocol (LDAP) is a directory service protocol that runs on a layer above the TCP/IP stack. It provides a mechanism used to connect to, search, and modify Internet directories. The LDAP directory service is based on a client-server model. The function of LDAP is to enable access to an existing directory". Thanks to its structure of directories, LDAP can provide operation for users' authentication and authorization. In fact, LDAP stores additional information about a user that can be retrieved during the authorization phases. Moreover, LDAP provides operations needed to interact with the directory-server database. Example of operations are: add, which creates a new entity in the database, and modify that changes some parameters of a particular entity. For authentication, a special bind operation is provided. The C3ISP Development and Test Bed environments provide an LDAP service that could be used for such purposes as well (see 10.1.2). Since the LDAP communication between clients and server is not encrypted, the transferred data are not protected (even if a client authentication has required to establish a connection with the LDAP server), LDAPS (LDAP over Secure Socket Layer) is recommended for securing the exchange of data over the LDAP since the data exchanged are encrypted by different cipher suites supported by the TLS (Transportation Layer Security) protocol [69].

- **OAuth2** [57]: it is an open standard to authorization. It provides client applications a "secure delegated access" to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. This is obtained through the use of single token that are released for a specific set of authorisations requested by the entity that wants to provide the service. However, the token is released only after a correct authentication of the resource-owner or end-user. Nowadays, OAuth is often used by websites that wish to provide access to their services using users' credentials on well-known providers such as Google, Facebook or Twitter.
- **OpenID** [61]: it is an open standard and decentralized protocol by the non-profit OpenID Foundation that allows users to be authenticated by certain co-operating sites (known as Relying Parties or RP) using a third-party service. In this way, the entity, which is charge of managing the authentication system, does not need to build up an ad hoc system. So, users can log into multiple unrelated websites without having to create single account with their information over and over again. Basically, users create a single OpenID account and then use those accounts to sign onto any website which accepts OpenID authentication. The OpenID standard provides a framework for communications that must take place between the identity provider and the OpenID acceptor (the "relying party"). The OpenID protocol does not rely on a central authority to authenticate a user's identity.

7.2. Key and Encryption Manager

Key and Encryption Manager is responsible for key management and encryption services needed for ensuring the confidentiality of the shared data across the C3ISP architecture components. In fact, the C3ISP data bundle, described in 4.1, is a cryptographic container built using these services, whereas Homomorphic Encryption is a technique used to preserve confidentiality even during computational services (more later).

This component leverages cryptography techniques which include solutions to encrypt data, a mechanism that can be used to protect user's privacy and confidentiality in general. The encryption is a cryptographic process to transform a message or plaintext into a *ciphertext* using a certain algorithm based on an encryption scheme. The process is reversible: the obtained ciphertext can be transformed into plaintext if decrypted, knowing the key used in the encryption scheme. In fact, in cryptography, encryption and decryption processes use mathematic objects called keys.

There are two kinds of cryptography concepts that we will cite and describe in the next sections:

1. Symmetric cryptography (also called secret-key cryptography);
2. Asymmetric cryptography (also called public-key cryptography).

Public key cryptography is often used in key management (for instance to distribute secret keys) and private key cryptography is employed for encryption since it is computationally more efficient.

In this section, we propose a design for the **Key and Encryption Manager** component (K&E Manager for short). K&E Manager secures, stores, and tightly controls access to tokens, passwords, certificates, API keys, and encryption services. K&E Manager handles key revocation, key rolling, and auditing (by integrating with the Secure Audit Manager, see 7.3) to trace any request towards the K&E Manager modules. Through a unified API, the other C3ISP subsystems can access an encrypted Key/Value store and network encryption-as-a-service to generate secrets. To address the C3ISP architecture protection needs, we specialise the K&E Manager into two main concrete modules namely Data Protected Object (DPO) – Key & Encryption Manager and Full Homomorphic Encryption (FHE) – Key & Encryption Manager:

- **DPO – Key & Encryption Manager** will provide services for Key management and Encryption tools. This module is dedicated to securing the C3ISP data bundle (see the Bundle Manager description in 4.1.4). Effectively, it provides a management for secret keys with respect to *symmetric-key technologies* and encrypt it using the Data Protected Object (DPO, i.e. the union of the CTI Data with the DSA Policies), thus allowing the Bundle Manager to form the C3ISP data bundle.
- **FHE – Key & Encryption Manager** will offer services for FHE keys management, including public & private key, *evaluation key*, and *transcryption key* (also called *trans-ciphering key*, more next). Evaluation key corresponds to the algorithm by which the data will be processed (it is a FHE specific object and it is described next).

With respect to the C3ISP high-level architecture defined in Figure 2, the Key & Encryption Manager interacts with the C3ISP subsystems and components as in the following figure:

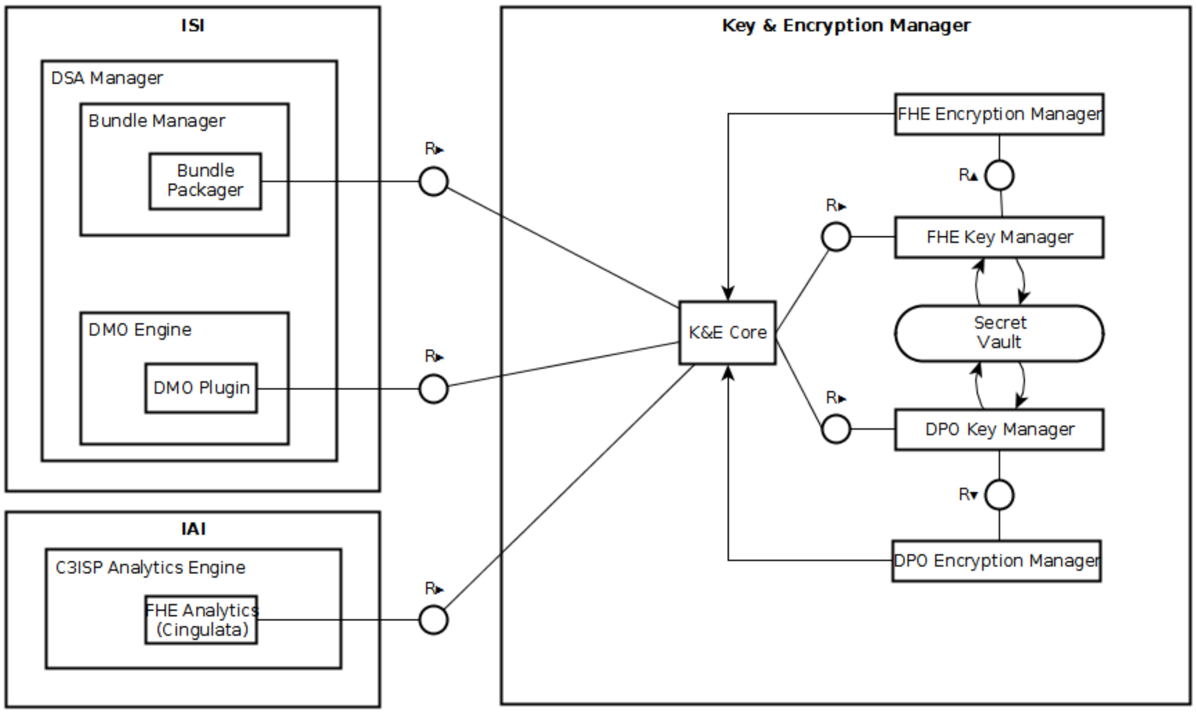


Figure 25: Key & Encryption Manager architecture

7.2.1. K&E Core

K&E Core is the centre of services of K&E Manager which allows 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). At high-level, the K&E Core exposes APIs for retrieving keys or invoking encryption services. It also provides API protection (i.e. a security layer to protect these APIs).

There are three scenarios where the ISI and IAI interacts with the K&E Manager via the K&E Core:

- The first one is the request from the Bundle Manager to encrypt the C3ISP data bundle, at CTI creation time;
- The second one is the request from the DMO Engine for retrieving the key (a public data, see *evaluation key* in 7.2.4.2) used to encrypted the CTI data with an homomorphic-friendly symmetric encryption algorithm;
- The final one is the request for HE computation from the FHE Analytics module of IAI.

7.2.2. Key Management

Key Management is a fundamental part of cryptographic technology and it refers to "*the process of handling and controlling cryptographic keys and related material during their life cycle in a cryptographic system (...)*" [88]. Key management is considered one of the most difficult aspects associated with the use of cryptography [38] [39] [40]; of particular concern are the scalability of the methods used to distribute keys and the usability of these methods.

Key life phases [65] are:

1. Key establishment;
2. Key storage;
3. Key destruction.

Key establishment (also called **key exchange**) is the process by which two (or more) parties establish a shared secret key, called the session key (in opposition to long-lived keys, also called long-termed key). The session key is subsequently used to achieve some cryptographic goal, such as privacy.

There are two kinds of key exchange protocol:

1. **Key transport** (also called **key distribution**) protocols in which a key is created by one party during **key generation** and securely transmitted to the second party. It can be done with:
   - a physically secure channel;
   - an interactive protocol over an existing encryption channel with a trusted third-party;
2. **Key agreement** protocols, in which both parties contribute information which jointly establish the shared secret key. Using a key-agreement protocol avoids some of the key distribution problems. It can done over an insecure channel.

Point to point communications and **centralized key management** are key distribution models [36] relevant to symmetric cryptography.

Elliptic Curve Diffie-Hellman (ECDH) is a key agreement protocol which permits to share a secret key. It uses public-key elliptic curve cryptography. The key distribution of public keys is done through public key servers. The protocol output is a secret key which serves to encrypt data with private-key cryptosystem. ECDH can be used with the elliptic curve Curve25519 which is fast and patent-free (https://cr.yp.to/ecdh.html), rather than Curve P-256 (from NIST). This curve is designed for offering 128-bits security level and it is widely used. Let us mention, a security vulnerability against another key agreement protocol called Diffie-Hellman (DH) discovered in May 2015. After the Logjam attack[16], secure use of DH requires doubling the size of the parameter needed to generate the secret key to use this protocol in a safe way. Vulnerabilities can always be discovered and so it is required to think about the scalability of such parameters.

It is a good practice to change key regularly: the maximal recommended lifetime of a key is called **cryptoperiod**. Cryptoperiods best-practices are indicated on the website https://www.keylength.com/, which also contains key size recommendations from national and internationally recognized standards organizations. During the cryptoperiod, keys have to be stored. **Key storage** can be centralised in one secure place. The security of key storage strongly depends on the security of the operating system in use. There also exists decentralised and distributed solutions that are secret-sharing protocols in which several parties take part. After the cryptoperiod, key have to be destroyed. **Key destruction** is not as simple as it seems. Once again, this depends on the operating system in use.

One solution to manage keys safely is the use of a **Key Management System (KMS)**. It is an integrated approach for generating, distributing and managing cryptographic keys for devices and applications. It is tailored to specific use-cases. It includes the backend functionality for key generation, distribution, and replacement as well as the client functionality for injecting keys, storing and managing keys on devices. In Figure 25, the KMS function is carried out by the DPO/FHE Key Manager and the **Secret Vault** (i.e. the keys store).

Cryptographic keys (also known as *Data Encryption Keys (DEK)*) are used to encrypt the data. A KMS serves to:

1. Generate DEK using a process involving a random number generator;
2. Securely store DEK using a Key Encryption Key (KEK);
3. Retrieve DEK using a secure protocol, such as TLS.

Examples of open source key management systems include:

- Barbican[17], the OpenStack™[18] security API (written in Python, Apache 2.0 license);
- Keybox[19], web-based SSH access and key management from Sean Skavanagh[20] (written in Java, Apache 2.0 license);
- Vault[21], secret server from HashiCorp[22] (written in Go, Mozilla Public License 2.0).

We favour an open source, well documented solution with software maintenance. On one hand, Barbican has already been used by HPE partner in the past and it is written in Python language. On the other hand, Vault seems to be a popular and stable solution. The KMS should be extendable. Indeed, homomorphic cryptosystems do not yet propose a standardized KMS. We plan to evaluate both solutions in particular with respect to software fault, incompatibility and maintainability.

[16] https://weakdh.org/

[17] https://wiki.openstack.org/wiki/Barbican/

[18] https://en.wikipedia.org/wiki/OpenStack

[19] http://sshkeybox.com/features.html

[20] https://github.com/skavanagh

[21] https://www.vaultproject.io/intro/index.html

[22] https://en.wikipedia.org/wiki/HashiCorp

7.2.3. DPO - Key & Encryption Manager

The C3ISP data bundle is a cryptographic container built using the services of the Data Protected Object (DPO) - Key & Encryption Manager (DPO K&E Manager, for short), which mainly consists of the DPO Key Manager and the DPO Encryption Manager modules (see Figure 25).

We give two scenarios where the **DPO Key Manager** module is used, that are aimed to address the Pilot use cases with different trust assumptions. Then, we discuss the **DPO Encryption Manager**.

One Key One Pilot scenario

In this setting, a symmetric key is created for each Pilot by the DPO Key Manager after request. It is the simplest case in terms of management with the fewest keys. But this requires that each Prosumer in a Pilot trusts other Prosumers. Indeed, all Prosumers' data will be encrypted with the same symmetric key.

One Key One Prosumer scenario

A second case can be considered where for each Prosumer in a Pilot a symmetric key is created by the DPO Key Manager. It removes the trust assumption of the first scenario. In this case, each Prosumer's data will be encrypted with its own symmetric key.

In each scenario, the DPO K&E Manager interacts with the Bundle Manager as follows:

1. Bundle Manager invokes a request for encrypting DPO (union of CTI Data and the DSA Policies) via an API of K&E Core;
2. K&E Core contacts the DPO Key Manager for retrieving the corresponding symmetric key of the pilot/Prosumer;
3. DPO Key Manager invokes DPO Encryption Manager to encrypt the DPO with the selected symmetric key;
4. The encrypted DPO is returned to Bundle Manager via K&E Core Manager.

7.2.3.1. Symmetric Encryption

In the C3ISP Framework, Data Encryption Keys (DEK) are used to encrypt C3ISP data bundles and also for Data Manipulation Operations (see next section for encryption via DMOs for HE computation). In this section, we describe the encryption process and a standard solution to encrypt data in the first context.

Encryption protects information and it permits confidentiality (secret communication) between communicating parties. The fastest solution is **symmetric cryptography**. It offers computational security, that is, it is secure assuming adversaries are computationally limited. In symmetric cryptography, **symmetric key** is used to encrypt and decrypt ciphertext respectively by sender and receiver and it is a secret. Beforehand, this requires to generate and share a secret key for each couple of communicating parties. In fact, one disadvantage of symmetric cryptography [36], is that there are numerous keys to manage in a large network. Moreover, secret key must remain secret at each end. It is a good practice to change keys regularly (see considerations about cryptoperiod and sharing protocols in 7.2.2). Once a secret key is shared between two actors, it can be used to encrypt messages and decrypt ciphertexts.

The **Advanced Encryption System** (AES for short) is a well-known (2001) and standardized cryptosystem [34] by the National Institute of Standard and Technologies (NIST) of the US. It is a symmetric block-cipher which operates on 128-bit blocks (it is the only block size specified in the AES standard). AES can be used with 128-bit keys (it is called AES-128), this minimal key size should be convenient to ensure security objectives in C3ISP context, that is 128 bits of security (at least 5 years of computation is required to break it for the most pessimistic). It is fast both in software and hardware. AES-128 is an adapted solution to encrypt a 128-bit message and it is a deterministic algorithm. In practice, when we encrypt a variable-length message, this requires to employ a block cipher mode of operation to offer confidentiality with a repeated use: AES-128 uses a block-cipher encryption mode. We can consider a **mode of operation** with an **initialization vector** (IV), which is a public data which permits to randomize encryption. In **probabilistic encryption (**also called randomized encryption**)**, multiple encryptions of the same plaintext with the same key produce distinct ciphertexts. For example, a popular family of modes of operation is **authenticated encryption mode**: it offers confidentiality and authentication. In this family of modes, GCM (Galois Counter Mode) and CCM (Counter with CBC-MAC) are patent-free authenticated encryption modes. The NSA Suite B Cryptography uses AES with GCM mode in RFC (request for comments) concerning SSH, IPsec and TLS cryptographic network protocols [44] [45] [46]. Concerning the IV, its description depends on the chosen mode. For GCM mode, guidelines to select IV are furnished in [41].

At a lower level, there are open source implementations for encryption services:

- Network Security Services (NSS) (written in C, assembly, free software, FIPS 140-2 [49], latest release June 2017) Transport Layer Security (TLS cryptographic network protocol);
- stunnel (written in C, cross-platform, free software, FIPS 140-2 [49], latest release April 2017) (TLS cryptographic network protocol);
- strongSwan (written in C, cross-platform, free software, latest release November 2016) Internet Protocol Security (IPsec cryptographic network protocol);
- OpenSSH (written in C, cross-platform, free software, latest release March 2017) Secure Shell (SSH cryptographic network protocol);
- OpenSSL (written in C, assembly, dual-licensed, latest release May 2017);
- Libgcrypt (written in C, free software, FIPS 140-2 [49], latest release June 2017, curve 25519) Key establishment library.

To encrypt the C3ISP data bundle, we will use the AES algorithm and we will evaluate these open source encryption services for the implementation.

7.2.4. FHE - Key & Encryption Manager

C3ISP data bundles are created as in the previous description (DPO K&E Manager), when the Prosumer's data is sent to the C3ISP Framework (see also Create CTI use case in section 8.1).

In contrast, in this section, the Encryption Manager is used to operate homomorphic encryption, with or without transcryption operation. Homomorphic encryption and transcryption are introduced in the two next sections. From a high-level view, in C3ISP, homomorphic encryption uses an *asymmetric cryptosystem* and transcryption uses a *symmetric cryptosystem*. Both use non-standard cryptosystem contrarily to what we have just described in previous section, where the DPO K&E Manager uses the AES cryptosystem. This has to be taken into account when choosing the Key Management System (KMS).

FHE - Key & Encryption Manager interacts with the two family of cryptosystems: symmetric and asymmetric. Symmetric encryption has been introduced in previous section with DPO K&E Manager. Let us now introduce asymmetric cryptography. An asymmetric encryption scheme requires two keys instead of the single key used in symmetric cryptography: a **public key** to encrypt data and a **private key** to decrypt data. The private key is secret like the symmetric key is in symmetric cryptography. The public key is not secret. Key management is simplified in asymmetric cryptography since the private key is not shared. Nevertheless, authenticity of public key needs to be ensured before encrypting data.

We consider a scenario with producer(s), the C3ISP Analytics Engine and consumer(s). If there are several consumers, the process described in this section, key distribution and data encryption has to be repeated for each consumer.

7.2.4.1. Homomorphic Encryption

Homomorphic Cryptography offers solutions to do *analytics* on ciphertexts without decrypting (see D7.1, section 2.2.1.1). It permits for instance to delegate computations to an untrusted server (e.g. the C3ISP Analytics Engine/FHE Analytics in C3ISP Framework): homomorphic computation is briefly explained in Appendix 2: Homomorphic Computation.

The **Fan-Vercauteren scheme** [35] (FV for short) is a public-key homomorphic encryption scheme (2012). Security in public-key cryptography is often based on the difficulty of a mathematical problem. FV security is based on the Ring variant of the Learning With Errors problem (RLWE) introduced in 2010 [47]. This problem reduces to an old and difficult mathematical problem: the shortest vector problem over (ideal) lattices. Homomorphic schemes such as FV employs the notion of (ciphertext) noise to guarantee the security. It can be viewed as an error introduced deliberately to encrypt data as in code-based cryptography. Noise is sampled from a distribution; it cannot be computed exactly. The size of the noise has to be minimized. Indeed, decryption is not correct if the size of the noise is above a threshold. During encryption, noise is added to compute a ciphertext.

<u>We say a ciphertext is **fresh** if it is obtained directly from a plaintext, and **homomorphic** if it is obtained after homomorphic operations.</u>

A fresh ciphertext contains a small size noise. Homomorphic operations such as (homomorphic) multiplication increase size of the noise. When homomorphic operations are done, ciphertext size increases. A fresh ciphertext is always smaller than a homomorphic ciphertext. In FV, ciphertexts are represented by polynomials. Sizes are computed using the maximal absolute value in the polynomial. Polynomial degree and coefficient size are additional parameters. These parameters permit trade-offs between security, time and memory usage, and the number of possible operations over ciphertexts. On the other hand, security analysis is complex and there is no key size recommendation contrarily to standardised cryptography. To simplify analysis, some parameters have to be fixed.

A homomorphic protocol can be divided into three computational parts:

1. The encoded data is encrypted by clients with the public key of final client;
2. The server evaluates a Boolean circuit over fresh ciphertexts, which correspond to an analytic treatment on data;
3. Finally, the client decrypts the resulting ciphertext with its private key.

As numerous RLWE-based homomorphic schemes, FV scheme employs a third key besides the public key and the private key. It is called the **evaluation key** (or relinearisation key). It is a public data. It is used after having operated the homomorphic multiplications. In these schemes, homomorphic multiplication is followed by a procedure of relinearisation on the ciphertext, which decreases the ciphertext size as well as the noise growth which depends exponentially on the ciphertext size. In the **Cingulata** tool [11] used for the FHE Analytics module (see 5.1.1), it occurs after each homomorphic multiplication. The public key is used by the client to encrypt data, in contrast the evaluation key is used by the server. If key size is not a constraint for the client, evaluation key can be part of the public key. Each consumer has one private key and one public key associated with the homomorphic cryptosystem.

Cingulata offers a compiler and runtime environment that permits to evaluate homomorphic computations by taking C++ code as input.

In FHE analysis scenario, the FHE K&E Manager interacts with the FHE Analytics as follows:

1. FHE Analytics sends a request to the K&E Core for retrieving the evaluation key corresponding to an analysis method via the FHE Key Manager;
2. FHE Key Manager returns this key to FHE Analytics via K&E Core, before the Cingulata service in FHE Analytics starts to perform the analytics on the encrypted CTI data in HE format;
3. The FHE Analytics result is sent to the C3ISP Framework via the ISI API create method (Create CTI).

7.2.4.2. Homomorphic Encryption with Transcryption

Transcryption enables to decrease the communication cost, between clients (producers) and server (in our case the C3ISP Analytics Engine). It helps the producers, but it has an additional server-side cost. It is a hybrid encryption technique, which employs two cryptosystems A and B. This technique permits to transform an encryption of a message with the first cryptosystem A to an encryption of the same message with the second cryptosystem B. The cryptosystem A is a **symmetric homomorphic-friendly cryptosystem** (it will be defined in the next paragraphs). The cryptosystem B is a **homomorphic public-key cryptosystem** (as described in the previous section), which permits to evaluate circuits over data encrypted with B. Transcryption adds symmetric keys shared by producers and consumer in the homomorphic protocol. Let us consider a case with only three actors: one producer, one C3ISP Analytics Engine and one consumer.

The producer task is easier with transcryption during first part of communication. The producer uses cryptosystem A[23] which is fast and obtains a small ciphertexts that it sends to the C3ISP Analytics Engine for computation. At the end, the consumer decrypts a homomorphic ciphertext obtained with cryptosystem B[24].

However, the C3ISP Analytics Engine task is heavier with transcription. It can be decomposed like this:

1. C3ISP Analytics Engine receives encrypted data with symmetric key in cryptosystem A;
2. It first encrypts it a second time with public key in cryptosystem B;
3. It evaluates a decryption circuit of cryptosystem A on twice-encrypted ciphertexts with encrypted symmetric key with B;
4. It obtains encrypted data with B of initial data – we call it **transcrypted data**;
5. It evaluates the circuit which corresponds to the desired analytics operation (also called treatment);
6. It submits the result to the ISI for storing it.

[23] More precisely, the cryptosystem A is applied thanks to a DMO specified in the DSA policies when the producer submits its CTI data to the C3ISP Framework. Then when a consumer asks for a FHE Analytics to the C3ISP Analytics Engine, the CTI data encrypted with A is used.

[24] This means that the result of the FHE Analytics is stored as an encrypted CTI data with cryptosystem B. When a consumer needs to retrieve it from C3ISP, then it must be decrypted with cryptosystem B.

To explain it better, we decomposed the circuit in two: decryption of A and analytics treatment. In practice, these are two sub-circuits of one circuit. The multiplicative depth of this circuit is the sum of the multiplicative depths of the two sub-circuits. A homomorphic-friendly cryptosystem should have a circuit of decryption of small multiplicative depth: however, this is not the case for the well-known AES cryptosystem, whose decryption circuit takes 18 minutes according to [49] for a standard security level. Let us focus on the first sub-circuit. Each producer has one symmetric key, that is one symmetric key associated with the symmetric cryptosystem used for transcryption. Each consumer has one private key and one public key associated with the homomorphic cryptosystem. For ease, consider only one producer and thus only one symmetric key. The producer encrypts each CTI data with its symmetric key.

In the C3ISP Framework, we will encrypt the CTI data with the cryptosystem A via a Data Manipulation Operation (DMO) that is invoked by the DMO Engine (see 4.1.3) when needed (i.e. when specified in the DSA policy).

The FHE Key & Encryption Manager interacts with the DMO Engine as follows:

1. DMO Engine sends a request to encrypt CTI data with cryptosystem A to the FHE K&E Manager;
2. K&E Core selects the private key for cryptosystem A by asking the FHE Key Manager, and then asks FHE Encryption Manager to encrypt it with cryptosystem A;
3. K&E Core returns the encrypted CTI data to the DMO Engine.

The FHE Key & Encryption Manager interacts with the C3ISP Analytics Engine as follows:

1. The C3ISP Analytics Engine sends encrypted CTI data to the FHE K&E Manager;
2. K&E Core selects the key from FHE - Key Manager, which retrieves the (stored) homomorphic encryption of the symmetric key, and then asks for transcrypting to the FHE – Encryption Manager;
3. The transcrypting is processing now running: the FHE Encryption Manager re-encrypts the encrypted CTI data with each public key;
4. The FHE – Encryption Manager finishes to perform the transcrypting process and returns back to the C3ISP Analytics Engine (FHE Analytics) the transcrypted CTI data via K&E Core;
5. Finally the FHE Analytics perform its computation and the C3ISP Analytics Engine saves the resulting CTI data into the C3ISP Framework via the ISI (createCTI method).

Let us finally introduce a solution for transcryption. **Kreyvium** [16] is a homomorphic-friendly symmetric cryptosystem with a 128-bit key to ensure a security level of 128-bit. It is a variant of Trivium [51] cryptosystem (2008). It is a stream cipher contrarily to AES which is a block cipher. These are the two families of symmetric cryptosystem: in stream cipher, plaintext is encrypted bitwise rather than block-wise and it uses the notion of **keystream**, that is, a stream of pseudo-random data. Kreyvium is an additive IV-based stream cipher. If we consider a binary message, it means that encryption (resp. decryption) can be separated in the two following phases:

1. Keystream is generated from the symmetric key and the initialization vector;
2. Ciphertext (resp. plaintext) is obtain by *xoring* plaintext (resp. ciphertext) and keystream.

The generated keystream is <u>independent of the message</u> and it is generated during the **offline** phase, which is the costly phase (i.e. can be done once for all CTI data). The second phase depends on the message and it is the **online phase**, which is low-cost as it is only a XOR operation.

7.3. Secure Audit Manager

To satisfy the security requirement **C3ISP-Sec-106**[25], described in D7.1 section 2.2.1.3, we want to use a centralised audit log management system that allows collecting, storing, and analysing the received log information. All the C3ISP components should be integrated with the **Secure Audit Manager** to track all the critical operations (i.e. the policies evaluation and their enforcement results, analytics execution, etc.) in order to achieve the accountability requirement for the C3ISP Framework.

The accountability derives from auditing and identity (see section 7.1); it means that, leveraging on these services, C3ISP platform will be able to retroactively establish who did a certain action on data, when this event occurred and how the action was performed.

The "Secure" property of the Audit Manager means that it has to provide a list of features to preserve confidentiality, integrity and availability of audit logs. In particular, it should:

- Achieve confidentiality by means of secure transport mechanism from the log collection point to the log storage area;
- Achieve integrity by means of a tamper-proof log storage system (e.g. a write-only log database). Also, using digitally signed audit logs from the source to the Secure Audit Manager helps achieving this goal;
- Achieve availability by making sure logs are not lost either during transportation (e.g. by using unreliable transport protocols such as UDP, by using queue mechanism in case of failures or communication issues, etc.), or due to Secure Audit Manager fault (e.g. by using a replicated storage area).

The Secure Audit Manager should provide an analysis interface to securely access the stored logs, by assuring controlling access (to authorised people only), segregation (not all logs should be available to any user accessing it), secure access via a protected channel (e.g. TLS over HTTP), as well as a form of self-audit (i.e. trace who is using the Secure Audit Manager).

The Secure Audit Manager could be used for several purposes, including compliance validation and operational monitoring. In particular:

- Compliance validation, means that a Prosumer might want to be confident and validate that the CTI data has been used in compliance with his or her DSA policies;
- Operational monitoring, means that who is operating the C3ISP Framework might want to understand if it is running as expected.

The Secure Audit Manager should expose a standard interface to the C3ISP components, based on REST technology [32], to allow them to send the audit logs to the centralised storage. We plan to use an already existent Secure Audit Manager solution and we briefly evaluated some open source solution, including:

[25] C3ISP-Sec-106: "For accountability purposes, C3ISP has an auditing subsystem that traces the enforcement results of the policies"

- The Elastic Stack[26] (formerly ELK Stack): it is a well-known log management system that includes a distributed fault-tolerant NoSQL database (Elasticsearch), a data collection and log parsing (Logstash) and a visualisation platform (Kibana);
- Graylog[27]: it is a log management platform (log collection and parsing), also based on Elasticsearch, that includes a Web-based analysis and visualisation front-end.

However, the C3ISP Framework should be decoupled from the specific Secure Audit Manager solution and be able to integrate with other off-the-shelf products, including commercial solutions: using standard communication protocols and log formats will allow to meet this goal.

[26] https://www.elastic.co/products

[27] https://www.graylog.org/

8. Data Flow Diagrams

In this section, we describe the basic C3ISP Framework use cases by detailing the flow of information between the components and their responsibilities. Further, we describe how these basic use cases (i.e. operations) can be combined to realise complex scenarios. For each flow, a UML [33] sequence diagram illustrates the communication and the temporal order of the interactions.

The data involved in the flows can be either provided by the producer as CTI or the output of an analytics service (what we call 'result'). From the data flow point of view, there is no significant difference between them, because the result of an analytics operation can be considered a CTI itself. However, we will distinguish their nature at the DSA level. The DSA will provide the possibility to specify different rules for data (CTI) and analytics results.

8.1. Create CTI

This flow describes the operation that an external entity issues when it wants to provide input data to the C3ISP Framework.

The actor is a Prosumer in the role of a Producer. In this scenario, the Producer is intended as a Pilot application or the IAI on behalf of the analytics Pilot application, who wants to submit data to the C3ISP Framework. The objective of this operation is to share data and then asking for an analytics service (on it or on CTI shared by other Prosumers participating in the DSA).

The Producer submits the own information to the C3ISP Framework by invoking the **ISI API** which exposes a service (*Create Normalised CTI*) for creating a sharable protected CTI. This service is implemented in different steps which involve several C3ISP components:

- The **Format Adapter** to normalise the CTI by converting the data in a specific common format, adopted by the C3ISP Framework, as explained in D7.1 section 2.1.1. This means that the original data is expressed in the STIX standard format, enriched with additional fields for supporting the management of the data. This step is jumped if the input data are already in the expected format (for example, in case of log files). We foresee another implementation of the create method which does not use the **Format Adapter** and provide as input data a normalized CTI (*Create CTI*);
- The **DSA Adapter** for retrieving the suitable DSA from the DSA Manager, in order to have the rules to protect the CTI, and for creating the C3ISP data bundle, which represents the protected object (the CTI "plus" the DSA for protecting it). The bundle is stored in the **Data Protected Object Storage** by the DSA Adapter and an identifier is returned for retrieving the CTI in the further operations;
- The **Key & Encryption Manager**, which is responsible for the encryption of the CTI and it is invoked by the DSA Adapter, before creating the bundle.

The interaction between these components is shown in the following sequence diagrams: as we explained before, we support two different implementations of the create method: one for creating CTIs which need to be formatted before the creation, and one for CTI data which are already in the expected format. Unless the usage of the **Format Adapter**, the flows proceed analogously.

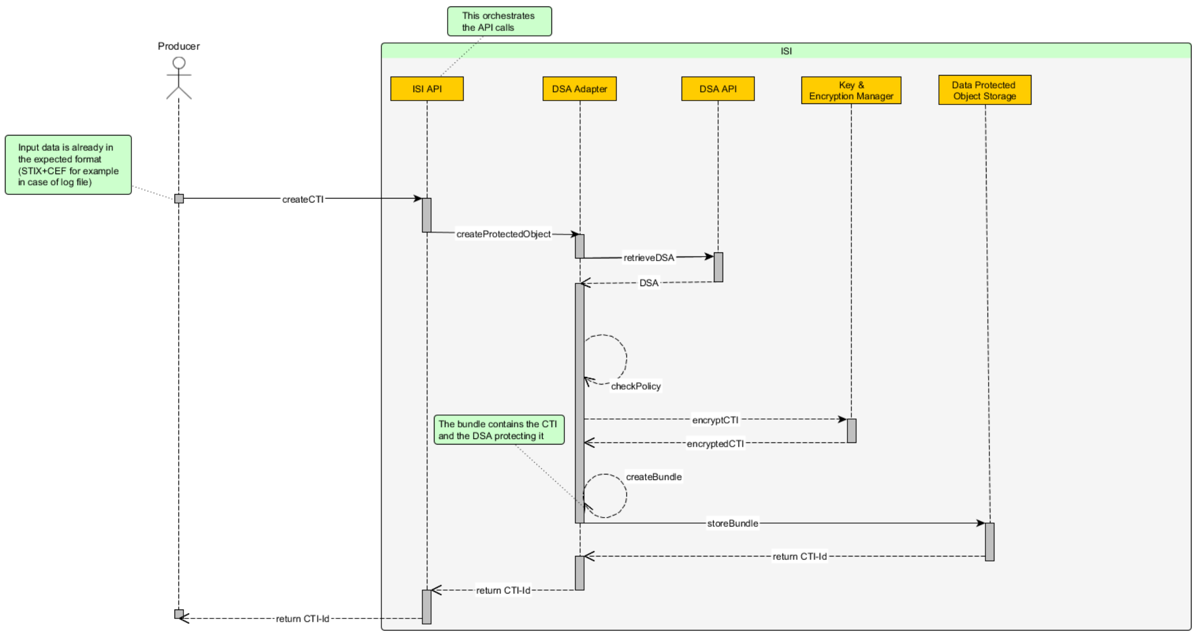

This orchestrates the API calls
Producer
ISI
ISI API
DSA Adapter
DSA API
Key & Encryption Manager
Data Protected Object Storage
Input data is already in the expected format (STIX+CEF for example in case of log file)
createCTI
createProtectedObject
retrieveDSA
DSA
checkPolicy
encryptCTI
encryptedCTI
The bundle contains the CTI and the DSA protecting it
createBundle
storeBundle
return CTI-Id
return CTI-Id
return CTI-Id

Figure 26: Create CTI Sequence Diagram

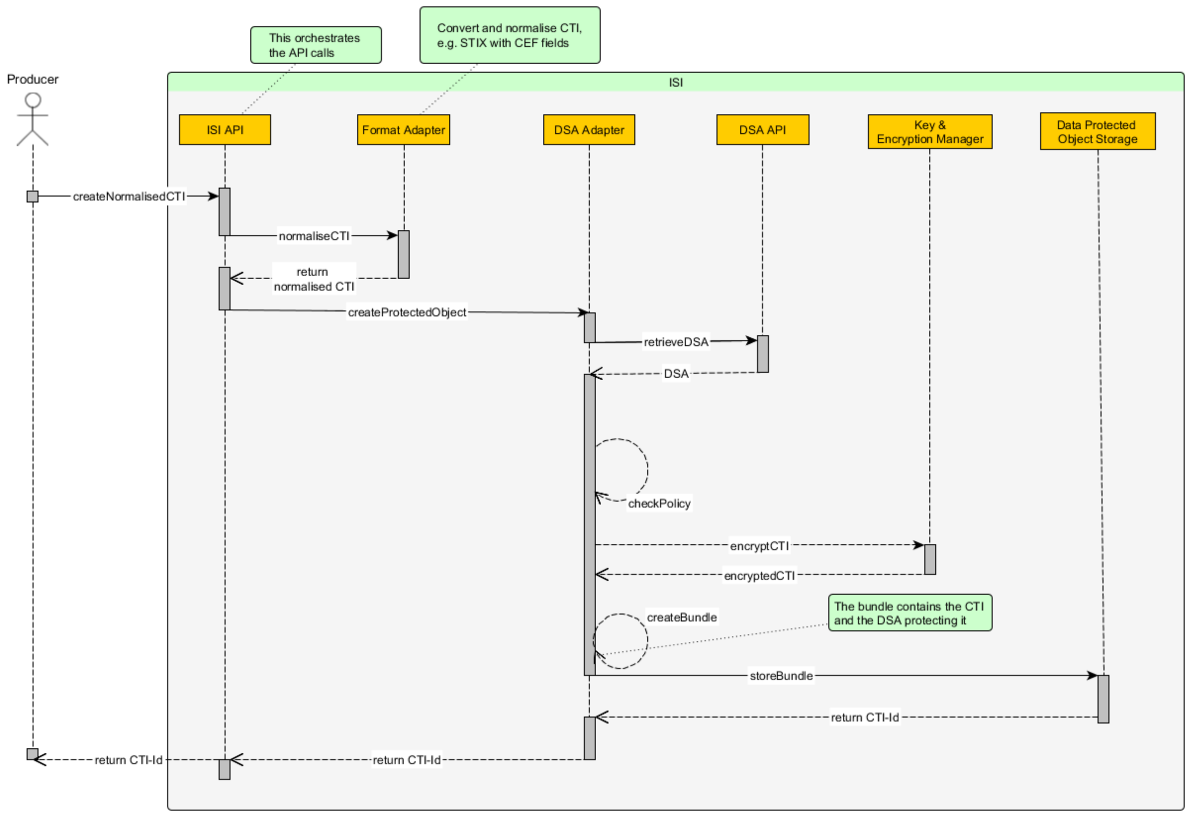


Figure 27 Create Normalised CTI Sequence Diagram

Note that to simplify the diagram we have omitted some validating steps at the DSA Adapter level. When the DSA is retrieved, it checks about the DSA validity since the DSA can expire, be revoked or someone might be updating it while it is going to be used (refer to D8.1 section

3.1 for the DSA state diagram). To further simplify, the interaction with the DSA Manager is limited to the DSA API and the inner DSA Manager behaviour is not shown.

The flow is applicable to all the deployment models described in Section 3, regardless of whether the ISI is local or remote.

8.2. Read shared CTI

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 rules.

The actor is a Prosumer in the role of a Consumer. In this scenario, the Consumer is intended as the Pilot application or the IAI on behalf of the analytics Pilot application, who wants to read a CTI already stored in the DPOS.

The actor must have a valid reference id to the CTI in order to invoke the *Read shared CTI* method exposed by the **ISI API**. Before getting the protected object from the **DPOS**, the policies defined in the attached DSA must be evaluated. As described in the *Create CTI* flow, the **DSA Adapter** is contacted for checking the DSA validity and, if it is valid, the DSA access and usage rules are processed. Only if the read operation on the required data is allowed for that Prosumer, the CTI is delivered to the requestor from the DPOS. Additionally, before providing CTI to the Prosumer, a data manipulation operation might be applied (for example, for obfuscate some CTI information), as prescribed in the defined DSA rules.

The sequence diagram, in Figure 28, shows the flow just described.

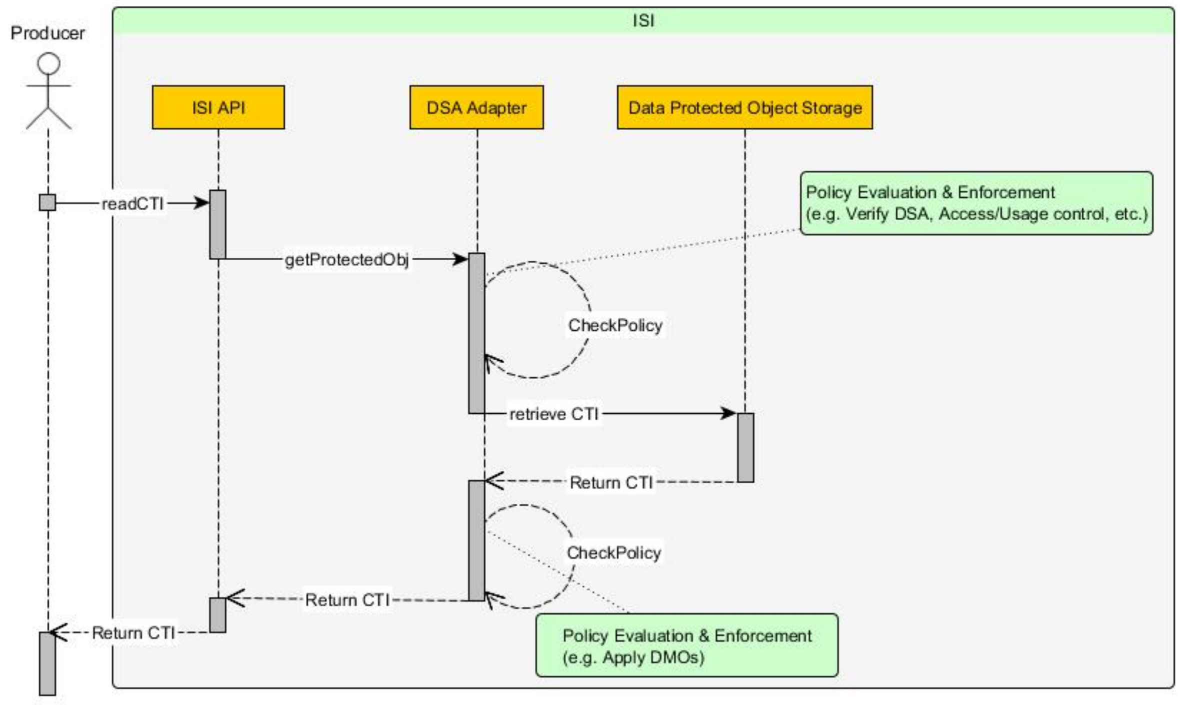

ISI
Producer
ISI API
DSA Adapter
Data Protected Object Storage
Policy Evaluation & Enforcement (e.g. Verify DSA, Access/Usage control, etc.)
readCTI
getProtectedObj
CheckPolicy
retrieve CTI
Return CTI
CheckPolicy
Return CTI
Return CTI
Policy Evaluation & Enforcement (e.g. Apply DMOs)

Figure 28: Read CTI Sequence Diagram

The flow is applicable to all the deployment models described in Section 3, regardless of whether the ISI is local or remote.

8.3. Move CTI

This flow describes the operation provided by the C3ISP Framework to move a previously "created" (see Section 8.1) object from an ISI node (source) to another ISI node (target). This is realised as a create operation on the target node with the same object retrieved from the source node, followed by a delete operation of the source node.

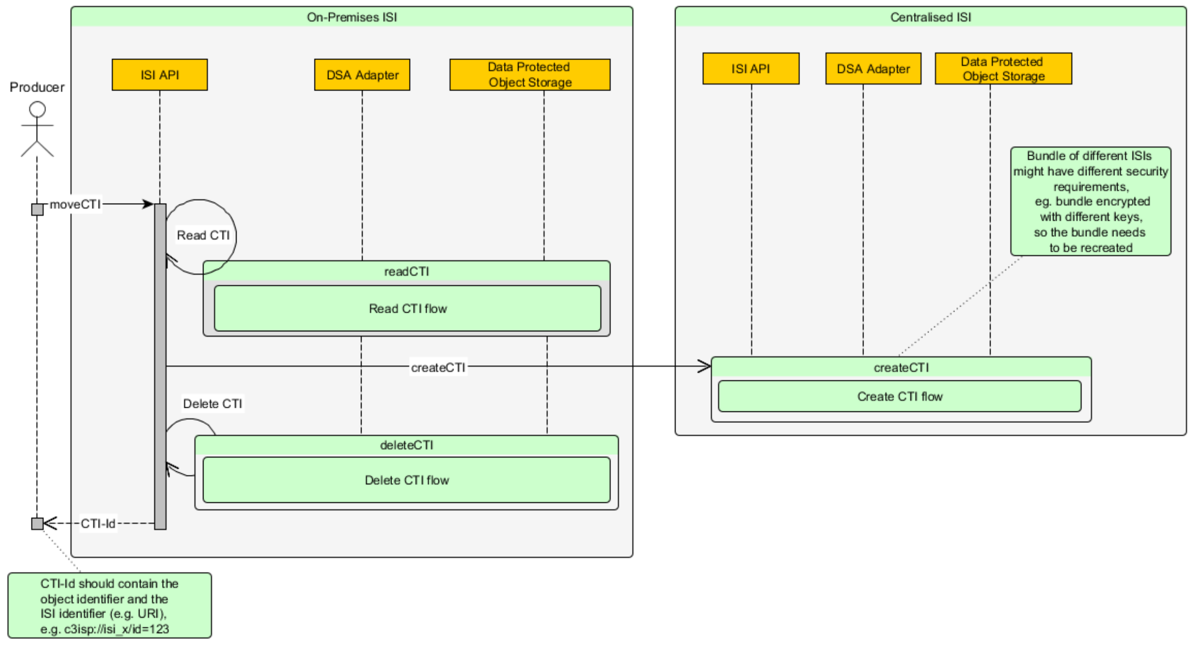
Typically the actor is a Prosumer in the role of a Producer. In this scenario, the Producer is intended as the Pilot application or the IAI on behalf of the analytics Pilot application, who wants to *Move CTI* from a local ISI (e.g. from the Producer premises) to a remote ISI (i.e. not under the control of the Producer) for both sharing and analysis. The producer invokes the *Move CTI* method exposed by the **ISI API**, giving a valid CTI id reference and a reference to the remote/target ISI, as input parameter.

Since the object is stored in the local ISI, the attached DSA policies must be evaluated, for example to check if the move operation is allowed and if there are DMOs that should be applied before moving. Once the policies at the local ISI level are enforced, the CTI is retrieved from the local **DPOS** (potentially manipulated according to the policies) and then created at the remote target ISI (via the remote *Create CTI*). At this stage, other policies are checked, e.g. to verify that this operation is allowed and how.

It is important to note that in this way different policies can be applied locally or remotely in the ISI. In particular, optional DMOs might be applied before moving the CTI to the remote ISI, while other (different) DMOs might be applied when the CTI is stored into the remote target ISI.

For the same reason, before storing the new CTI, a new C3ISP data bundle is created (via the *Create CTI*), since a different set of policies should be applied to it in the remote ISI. The move operation returns to the Producer the id of the transferred CTI and, as side effect, the original CTI is deleted from the local ISI.

The sequence diagram of the scenario is represented in Figure 29.



On-Premises ISI
Centralised ISI
Producer
ISI API
DSA Adapter
Data Protected Object Storage
ISI API
DSA Adapter
Data Protected Object Storage
Bundle of different ISIs might have different security requirements, eg. bundle encrypted with different keys, so the bundle needs to be recreated
moveCTI
Read CTI
readCTI
Read CTI flow
createCTI
createCTI
Create CTI flow
Delete CTI
deleteCTI
Delete CTI flow
CTI-Id
CTI-Id should contain the object identifier and the ISI identifier (e.g. URI), e.g. c3isp://isi_x/id=123

Figure 29: Move CTI Sequence Diagram

The flow is applicable to a hybrid deployment model, where there is an on-premises ISI and a centralised ISI (see Section 3.2), so that the Producer wants to move a CTI from the local ISI to the remote one. The flow is analogous in a fully distributed deployment model (Section 3.4) for sharing results between ISIs.

8.4. Delete CTI

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

The actor is a Prosumer in the role of a Producer. In this scenario, the Producer is intended as a Pilot application or the IAI on behalf of the analytics Pilot application, who has the authority (i.e. the policies allow to perform this operation) for delete a CTI from the C3ISP Framework. The objective of this operation is remove data from the Data Protected Object Storege, to make the CTI unavailable to the C3ISP Framework anymore.

Given the CTI-Id for identifying an existing CTI in the C3ISP framework, the ISI API invokes the DSA Adapter for retrieving the DSA associated to the CTI and to check the policies in order to allow or deny the requested operation. If the Prosumer is authorised to perform the operation, the CTI (and the bundle) is deleted from the Data Protected Object Storage. The operation result is returned to the caller as notification.

The following diagram (Figure 30) shows the flow just described.

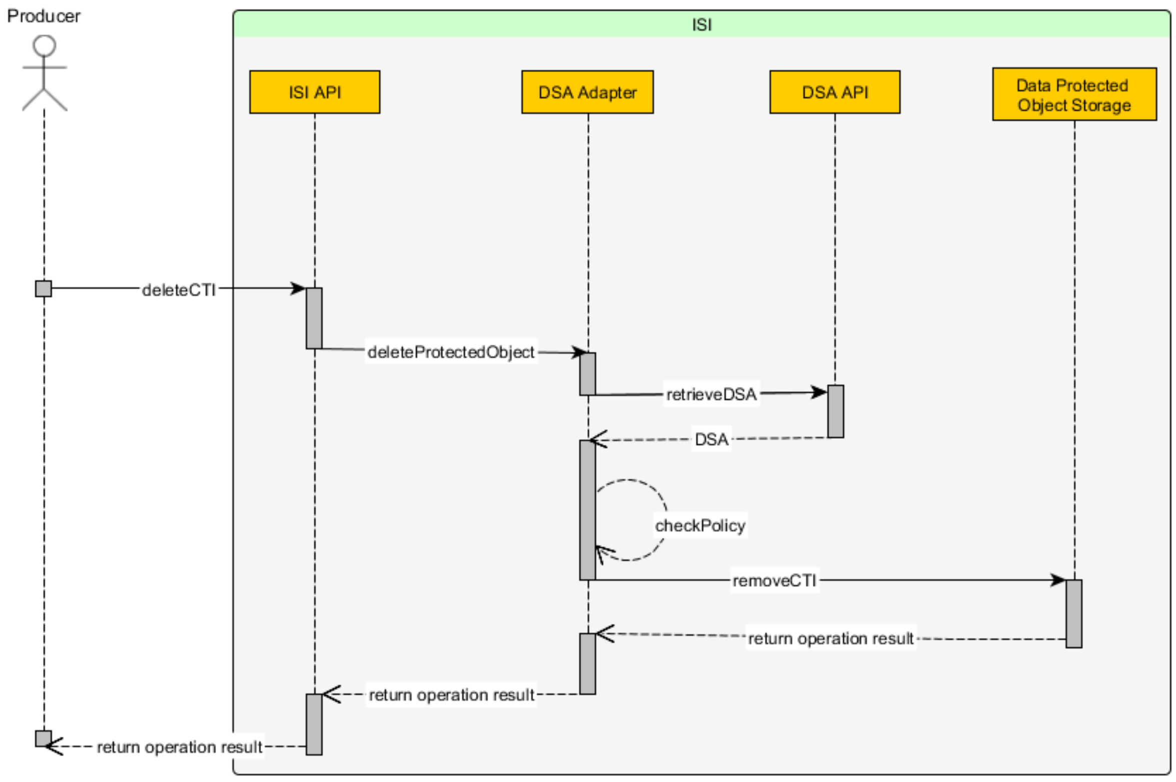


Figure 30: Delete CTI Sequence Diagram

8.5. Invoke C3ISP analytics service

This flow describes the feature provided by the C3ISP Framework for invoking a generic analytic service on a CTI, given the service name and the CTI-Id. The flow is also applicable in case of multiple analytics services invocation (see Section 5.5). The Consumer uses the suitable method exposed by the IAI API.

The implementation of this service involves:

- the ISI API, for retrieving the CTI using the *readCTI* operation;

- the Format Adapter, for retrieving data in the expected format;
- the C3ISP Analytics Engine, for performing the analytics operation leveraging on methods and tools offered by the C3ISP Framework for this purpose;
- the ISI API again, for creating the CTI as analytics result data.

The service returns a CTI-Id. For retrieving the content of the result the Prosumer should use the *readCTI* method of the ISI API, given the returned CTI-Id.

The following diagram shows the interaction between these components.

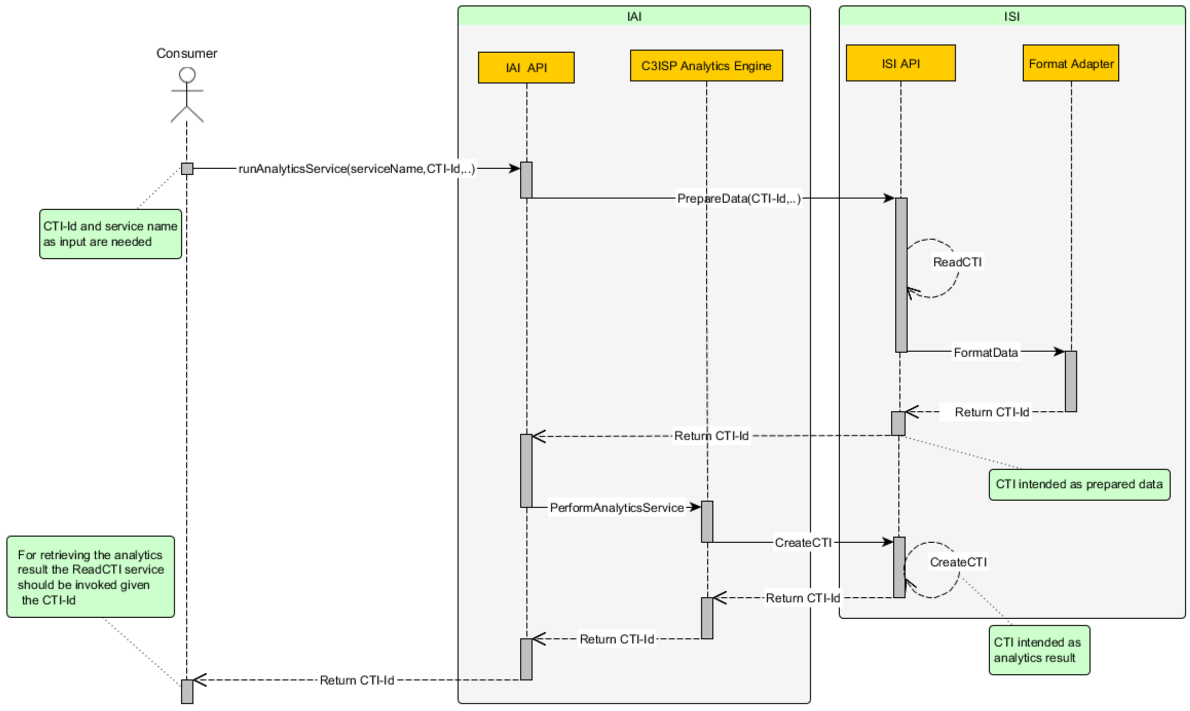

IAI
ISI
Consumer
IAI API
C3ISP Analytics Engine
ISI API
Format Adapter
runAnalyticsService(serviceName,CTI-Id,..)
CTI-Id and service name as input are needed
PrepareData(CTI-Id,..)
ReadCTI
FormatData
Return CTI-Id
Return CTI-Id
CTI intended as prepared data
PerformAnalyticsService
For retrieving the analytics result the ReadCTI service should be invoked given the CTI-Id
CreateCTI
CreateCTI
Return CTI-Id
Return CTI-Id
CTI intended as analytics result
Return CTI-Id

Figure 31: Invoke Analytics Service Sequence Diagram

8.6. *Invoke legacy analytics service*

In order to use the legacy analytics service that is exposed by the IAI API, the Prosumer is assumed to have a reference Id of the CTI that should later be consumed by the Legacy Analytics Engine (LAE). After the service request is received by the IAI API, it requests the CTI via the ISI API using the *readCTI* operation (see Section 8.2) and indicates that the CTI will be stored into a Virtual Data Lake (VDL) instance. The ISI API requests the Format Adapter to process and format the retrieved CTI in such a way that it is compliant with the associated DSA policy (e.g., encrypting and/or anonymising some portions of the data). The formatted data is sent back to the IAI API, which then instantiates the VDL instance (if not existent already), e.g. create a new HDFS folder, and stores the data into the VDL. The IAI API keeps hold of the access information to the VDL, such as, the connection details, access credentials, etc. Once the data is copied into the VDL, the IAI API sends a service request to the LAE, which then reads the data from the VDL and perform analytics on it. The VDL access details need to be communicated to the LAE in a secure and suitable way depending on available data source configuration options, e.g., the details may be included in the request message body or written into the LAE's system database directly. After the analytics operation is completed the result will be returned to the Prosumer via the IAI API. Figure 32 shows the sequence diagram of the described flow. In case the LAE requires direct user interaction for analysing the data, e.g. for visualisation purposes, the IAI API will create the VDL instance without specifically running an analytics job (see Section 5.5) and redirect the Prosumer's browser to the LAE user interface. Furthermore, the IAI API could also have both the synchronous and asynchronous API style, where it can notify the Prosumer once the analytics result is ready.

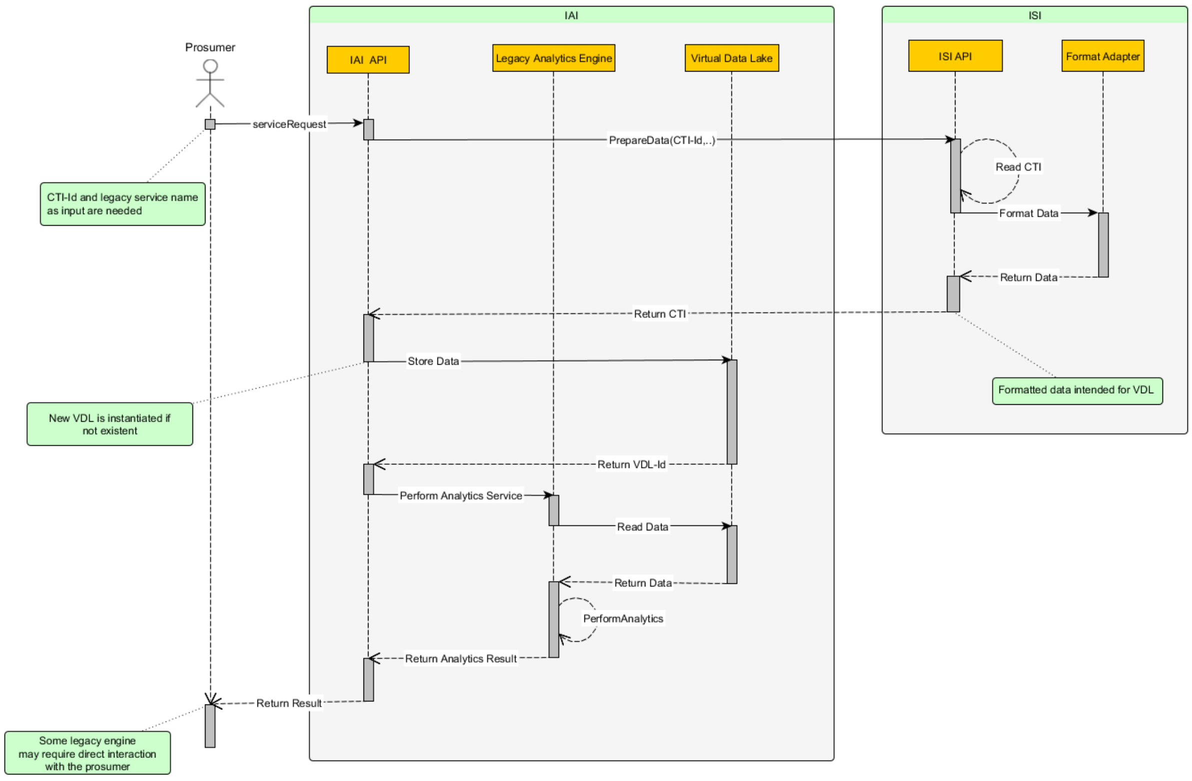


Figure 32: Invoke Legacy Analytics Service

9. Requirements mapping from D7.1

This section is dedicated to mapping the C3ISP reference architecture components with the framework requirements drawn in the D7.1 in order to check the coverage with them.

9.1. Data Sharing Requirements

Table 1 – Coverage of Data Sharing Requirements

ID	Requirements	Coverage
C3ISP-Fun-DS-001	C3ISP allows parties that want to exchange CTI data to define Data Sharing Agreements.	DSA Manager and its component DSA Editor. The Data Sharing Agreements are created and updated by the DSA Editor.
C3ISP-Fun-DS-002	C3ISP allows the sharing of files (including log data, threat intelligence data, analysis reports)	ISI subsystem, which is accessible by the ISI API. The shared data is stored in and retrieved from the Data Protected Object Storage.
C3ISP-Fun-DS-003	C3ISP grants Prosumers the control over the sharing of data (i.e. Prosumers have both tools and functionalities to specify "constraints" that regulate the data sharing process)	DSA Manager and its components allow Prosumers to control data sharing via the setting of constraint in the DSA policies using the DSA Editor.
C3ISP-Fun-DS-004	C3ISP allows controlling the process of data sharing at file level	The DSA Manager allows the DSA policy to be created, and the ISI subsystem allows the policy and data file to be stored together in the Data Protected Object Storage.
C3ISP-Fun-DS-005	C3ISP allows defining policies (i.e. a set of rules) that regulate the data sharing process	DSA Manager and its component DSA Editor. The Data Sharing Agreements are created and updated by the DSA Editor.
C3ISP-Fun-DS-006	C3ISP policies allow access control to the shared data (i.e. define conditions to be verified before accessing the data)	The DSA Adapter controls access to the Data Protected Object Storage by enforcing DSA policy stored with the data file.
C3ISP-Fun-DS-007	C3ISP policies allow usage control of the shared data (i.e. define conditions to be continuously verify while the data is being consumed and after it has been accessed)	The Service Usage Control Adapter is the component that enforces usage control on the shared data.
C3ISP-Fun-DS-008	C3ISP policies allow defining rules that can evaluate contextual information (i.e. information from the environment/use case)	The DSA Editor allows rules to be defined for contextual information.
C3ISP-Fun-DS-009	C3ISP allows defining notifications (i.e. email, SNMP, etc.) that are triggered once the analytic service result is available (i.e. be able to encode this requirement in a policy	Notifications are encoded in DSA policies as obligations when the analysis finishes. The Obligations Engine is a component of the Service Usage Control Adapter.

	rule). A notification mechanism could be email.	
C3ISP-Fun-DS-010	C3ISP provides evidences (e.g. audit logs) of the compliance to the sharing policies enforcement	The Secure Audit Manager from Common Security Service subsystem is responsible for storing auditing records.
C3ISP-Fun-DS-011	C3ISP policies allow writing “pre-processing rules” on the data to be shared, which are data manipulation operations performed before the data is shared with the other party(ies). These operations should include: (i) sanitisation operations (see 9.3) for minimising sensitive data exchange; (ii) encryption mechanisms	The pre-processing rules are encoded in the DSA policies as obligations to be carried out before data access is granted.
C3ISP-Fun-DS-012	C3ISP allows specifying policy rules to control the risk of data sharing (i.e. if a metrics is over a certain threshold, data can’t be shared or additional sanitisation measures must be applied before sharing)	The risk of data sharing will be encoded in the DSA policies. How this will be done will defined in the second year of the project.
C3ISP-Fun-DS-013	C3ISP could use an open and/or standard policy description language for data sharing (DSA/XACML)	The DSA Mapper converts from controlled natural language into XACML (with extensions).
C3ISP-Fun-DS-014	C3ISP allows defining multi-lateral Data Sharing Agreements, i.e. DSA between multiple Prosumers (two or more)	The DSA Manager supports defining the parties participating in the agreement.

9.2. *Data Analytics Requirements*

Table 2 – Coverage of Data Analytics Requirements

ID	Requirements	Coverage
C3ISP-Fun-DA-001	C3ISP allows defining policies (i.e. a set of rules) for data analytics operations to control what analysis can be performed on the Prosumer’s data	This will be an extension to the DSA policy to be provided in the second year of the project.
C3ISP-Fun-DA-002	C3ISP policies allows writing “post-processing rules” on analytics operation result, which are data manipulation operations performed before returning it to the Prosumer(s). DMOs should include data sanitisation and (de)encryption (see also C3ISP-Fun-DS-011 and 9.3)	This will be an extension to the DSA policy to be provided in the second year of the project.

C3ISP-Fun-DA-003	C3ISP provides an application programming interface for executing DMOs, such as privacy-preserving operations on data (e.g. data sanitisation or encryption)	The API for the DSA Adapter is provided by the DSA Adapter Front End via the ISI API.
C3ISP-Fun-DA-004	C3ISP allows executing privacy-preserving DMOs on all or part of the data	The DSA policy will allow data manipulation operations (DMOs) to be defined, and the DMO Engine will execute these.
C3ISP-Fun-DA-005	C3ISP provides an application programming interface for supporting the analysis on the data stored in C3ISP data lake, in compliance with the associated DSA policies	This is supported by the IAI API and ISI API.
C3ISP-Fun-DA-006	C3ISP provides an application programming interface to query data and analytics operation results that are stored in C3ISP data lake in compliance with the DSA policies	This is supported by the ISI API
C3ISP-Fun-DA-007	C3ISP supports standard query language (e.g. SQL) for querying data and analytics operation results from C3ISP data lake	C3ISP Analytics Engine (and maybe the DPOS) and Virtual Data Lake
C3ISP-Fun-DA-008	C3ISP provides a function for automatic threat classification of analytics operation results	*Not currently*
C3ISP-Fun-DA-009	C3ISP provides a function for automated mapping of analytics operation results to interested stakeholders/Prosumers that are specified in the DSA	*Not currently, but planned to be incorporate in the second year.*
C3ISP-Fun-DA-010	C3ISP provides a function to convert analytics operation results to standardised and machine-readable formats (e.g. STIX) in compliance with the DSA	The (ISI) Format Adapter supports the conversation of formats.
C3ISP-Fun-DA-011	C3ISP provides an application programming interface to integrate external analytics tools while preserving the policy compliance (i.e., extract data from C3ISP data lake and feed it into analytics tool)	The IAI API provides access to the Legacy Analytics Engine and Virtual Data Lake.

C3ISP-Fun-DA-012	C3ISP provides near real-time notifications of analytics operation results (see also C3ISP-Fun-DS-009)	The Obligations Engine of the Service Usage Control Adapter provides this functionality.
C3ISP-Fun-DA-013	C3ISP provides a function to query analytics operation results of specific categories (e.g. malware analysis, attack on cloud platform)	The IAI API for results stored in Virtual Data Lake. The ISI API for results stored DPOS.
C3ISP-Fun-DA-014	C3ISP supports different categories for analytics operations results, i.e. threat types, threat risks, threat origins, threat costs, regulatory and compliance concerns	The Legacy Analytics Engine for data stored in Virtual Data Lake. The C3ISP Analytics Engine for data stored DPOS.
C3ISP-Fun-DA-015	C3ISP supports the provisioning of analytics operation results in form of actionable items (e.g. security patches, recommended configurations, fixes, etc.). See also the OpenC2 description in 9.4.3.	The Legacy Analytics Engine for data stored in Virtual Data Lake. The C3ISP Analytics Engine for data stored DPOS.
C3ISP-Fun-DA-016	C3ISP provides a dashboard showing status and results of the analysis	It is not part of the C3ISP architecture, but a simple dashboard will be provided to the pilots during the second year.
C3ISP-Fun-DA-017	C3ISP allows scheduling of the provisioning of analytics operation results (e.g. on demand, periodical, etc.)	The DSA Policy and the Service Usage Control Adapter provide the scheduling.
C3ISP-Fun-DA-018	When using homomorphic encryption, before data analytics execution, data is represented as bits or integers.	The FHE DMO Engine will convert the data.
C3ISP-Fun-DA-019	When using homomorphic encryption, data is of constant length (in real world scenarios). If not, a possible solution is to compute a hash function (not necessarily a cryptographic one) on data.	The FHE DMO Engine will prepare the data.
C3ISP-Fun-DA-020	When using homomorphic encryption, analytics operands (cipher-texts) are encrypted bits (most current case) or encrypted integers with considered homomorphic cryptosystems.	The FHE DMO Engine will prepare the operands.
C3ISP-Fun-DA-021	When using homomorphic encryption, analytics operations are expressible in terms of two elementary operations:	The FHE DMO Engine will perform the homomorphic operations.

	(homomorphic) addition and (homomorphic) multiplication with considered homomorphic cryptosystems, Homomorphic sum (resp. product) of two cipher-texts is a cipher-text of the sum (resp. the product) of two associated plaintexts.	
C3ISP-Fun-DA-022	When using homomorphic encryption, analytics computation on encrypted bits is represented as a Boolean circuit with multiplicative depth[28] roughly 20 or 30.	The FHE DMO Engine will perform the homomorphic operations.
C3ISP-Fun-DA-023	When using homomorphic encryption, the number of multiplicative gates should be minimized to decrease latency of homomorphic evaluation.	This will be supported by the FHE DMO Engine.

9.3. Data Manipulation Operations

In the context of the DMOs described for C3ISP-Fun-DS-011 and C3ISP-Fun-DA-002, possible anonymization techniques that find application in the Pilots' use cases are:

- Suppression of identifiers (e.g. names);
- Generalization of values in certain finite domains (e.g. subnet masking);
- Randomization methods that anonymize individual values (and thereby one's membership in the data set) in such a way that accurate aggregates for certain functions (e.g. mean) can be produced when enough data is provided.

DMOs includes also homomorphic encryption described in 7.2.4.1.

9.4. Non-Functional Requirements

9.4.1. Information Security Requirements

Table 3 – Coverage of Information Security Requirements

ID	Requirements	Coverage
C3ISP-Sec-001	There is mutual authentication carried out between the C3ISP Framework and the Prosumers at the start of any communication.	The Identity Manager of CSS will support this.
C3ISP-Sec-002	Confidentiality and Integrity of the DSA communications between the	The Key & Encryption Manager of CSS will support this.

[28] Multiplicative depth is the maximum number of multiplicative gates between an input and an output of the circuit.

	Prosumers and C3ISP Service is guaranteed.	
C3ISP-Sec-003	The transfer of CTI from the Prosumers to the C3ISP framework is secure (w.r.t. confidentiality and integrity).	The Key & Encryption Manager of CSS will support this.
C3ISP-Sec-004	The integrity of the CTI data stored on the C3ISP Framework is maintained.	The ISI Data Protected Object Storage will support this.
C3ISP-Sec-005	The transfer of analysis results from the C3ISP framework to the Prosumers is secure (w.r.t. confidentiality and integrity).	The Key & Encryption Manager will support this.
C3ISP-Sec-006	C3ISP Framework is able to process anonymised or homomorphically encrypted CTI shared with it by the Prosumers.	The C3ISP Analytics Service will support the processing of anonymised CTI. The FHE DMO will support the processing of homomorphically encrypted CTI.
C3ISP-Sec-007	Minimum security level for FHE is at least 80 bits (security strength).	This is supported by the FHE DMO.
C3ISP-Sec-008	Maximum security level is at most 128 bits (computational efficiency), for real world scenarios.	This is supported by the FHE DMO.
C3ISP-Sec-009	The homomorphic encryption uses randomization methods (see section 2.1.3). It is required to have semantic security. That is, it should be hard to distinguish between the encryption of any two messages, even if the public key is known to the attacker and even if the two messages are chosen by the attacker (chosen plaintext attacks). (In return, cipher-text size is greater than plaintext size).	This is supported by the FHE DMO.

9.4.2. Regulatory Requirements

Table 4 – Coverage of Regulatory Requirements

ID	Requirements	Coverage
C3ISP-Sec-101	The Prosumers are given the details of their data's lifecycle at the C3ISP Framework (GDPR Requirement).	This is supported by the Security Auditor Manager of the CSS.
C3ISP-Sec-102	The Prosumers are able to reject or cancel the terms and conditions of	The DSA Editor will allow Prosumers to edit their policies at any time, and the Service Usage Controller Adapter will

	their DSA with the C3ISP Framework at any time. (GDPR Requirement)	implement changes to the DSA policies in near real-time.
C3ISP-Sec-103	The Prosumers are informed of any breach or compromise of the C3ISP framework within 72 hours, so that they can take remedial actions. (GDPR Requirement)	This is an operational matter.
C3ISP-Sec-104	The Prosumers are able to define data access and usage policies	The DSA Editor of the DSA Manager supports this.
C3ISP-Sec-105	Data sharing and data analysis is compliant with the law obligations and/or the industrial standard	It is possible to have pre-configured DSA templates ready with (some) GDPR rules to be used as a starting point to define specific Prosumers DSA policies.
C3ISP-Sec-106	For accountability purposes, C3ISP has an auditing subsystem that traces the enforcement results of the policies	The Secure Audit Manager provides this functionality.

9.4.3. Operational Requirements

9.4.3.1. Cloud Computing and Deployment Models Requirements

Table 5 – Coverage of Cloud Computing and Deployment Models Requirements

ID	Requirements	Coverage
C3ISP-Ope-001	C3ISP is available as a service, following the SaaS model	The services of the C3ISP Framework are provided via DSA API, ISI API and IAI API.
C3ISP-Ope-002	C3ISP is multi-tenant, where several tenants (i.e. pilots) can use the framework at the same time w/o troubles	This is to be covered by the implementation of the microservice architecture (e.g., see Section 2.1)
C3ISP-Ope-003	C3ISP is independent of the CSP where it runs (e.g. public or private)	The overall design makes the C3ISP Framework independent of any CSP.
C3ISP-Ope-004	DSA policies allow to specify different DMOs depending on the trust level the Prosumer has on the CSP or on other Prosumers in the federation	The DSA Editor of the DSA Manager support this.

9.4.3.2. Extensibility and Interoperability Requirements

Table 6 – Coverage of Extensibility and Interoperability Requirements

ID	Requirements	Coverage
C3ISP-Ope-101	C3ISP provides an open interface for application integration (e.g. a C3ISP API)	The services of the C3ISP Framework are provided via DSA API, ISI API and IAI API.
C3ISP-Ope-102	C3ISP uses a standard to represent data in order to simplify the integration with the framework	The Format Adapter will support standard formats, such as, STIX.
C3ISP-Ope-103	C3ISP should be able to represent different kind of *cyber observables*	This is supported because CybOX [7] are specific STIX objects.
C3ISP-Ope-104	C3ISP provides information related to the *cyber observables* which characterize it	This is supported because CybOX are specific STIX objects.

9.4.4. Performance Requirements

The performance of the C3ISP proof of concept implementation is not of primary concern.

Table 7 – Coverage of Performance Requirements

ID	Requirements
C3ISP-Per-001	C3ISP does not introduce significant delay when enforcing policies for sharing analytics operation results
C3ISP-Per-002	FHE services are deployed into a physical server, not into a Virtual Machine, because of using parallelism method for optimizing the Boolean circuits.
C3ISP-Per-003	With each pilot's use case, C3ISP defines an interval of tolerant response delay, in order to obtain a compromise resource availability for other requests on FHE services (*response delay requirement*). This requirement applies only if FHE is used.

9.4.5. Usability Requirements

Table 8 – Coverage of Usability Requirements

ID	Requirements	Coverage
C3ISP-Usa-001	C3ISP provides response information (results or errors) about requests (analytics query) to the C3ISP data lake (e.g. why the requested data cannot be provided to Prosumers)	
C3ISP-Usa-002	C3ISP provides a tool to guide and support the end user in the definition of DSA policies (authorisations, prohibitions, obligations)	This is supported by the DSA Editor,

C3ISP-Usa-003	C3ISP's processes are seamless and transparent in order to not interfere with the core operations of the pilots	This is a deployment issue, and can be solved by running C3ISP on separate systems.
C3ISP-Usa-004	C3ISP's representation of analytics results is effective and efficient for the end user.	It is not part of the C3ISP architecture, but a simple UI will be provided to the pilots during the second year.
C3ISP-Usa-005	To use the FHE technology, the decryption service (library) is installed or integrated in client applications.	This is an operational requirement for the testbed.

10. Update on the Development and Test Bed Environments

In deliverable D6.1, we planned to have two environments for supporting the C3ISP Framework implementation activities: Development and Test Bed environments. The Development Environment hosts all the tools to create, store, build and evaluate the quality and security of the software artefacts. For simplicity, we decided to have it deployed on a single virtual machine (VM).

The Test Bed Environment has instead been conceived to host a running instance of the C3ISP reference architecture, deployed to provide services to the Pilots. As described in Section 3, there are different deployment models for the C3ISP Framework. In our planned Test Bed will have the services centralised and will then be classified as Fully Centralised. However, since we designed the C3ISP Framework for multi-tenancy (or at least multi-user), we think it could be also used for the Hybrid mod, in which the Pilot will run its own local ISI instance.

The two environments are integrated: Development environment continuously delivers the software artefacts to the Test Bed environment once new or updated versions are available. In this way, the Pilots will always have the very latest release of the C3ISP Framework for their testing and evaluation activities. Having the latest software release could also incur in stability issues, but to mitigate this factor we already thought in deliverable D6.1 to adopt a continuous integration and delivery service that also performs automated testing, quality and security issue investigations.

The current installation will evolve over time in order to adapt to the evolution of the project, in case is needed. This is valid both for the basic infrastructure (OS and virtual machines configurations), as well as, for software and tools. Having a continuous integration service can also assure the issues that might rise for this kind of upgrades can be controlled by the reproducibility of the building and testing activities.

10.1. Development Environment

In the following sections, we describe what has currently been installed to match the requirements reported in deliverable D7.1 (sections 3.1 and 3.2)[29] for both operating environments. In particular, we split between base hardware/OS settings and software configuration.

All the environments described below are hosted at CNR data centre, and they can be reached through SSH for command line administration. We activated also secure HTTPS connections (HTTP over TLS) on each installed web service, by adopting digital certificates signed by “Let’s Encrypt”[30], a well-known free and open certification authority.

10.1.1. Base configuration

The base configuration of the development environment has been fully installed. It is based on a VM (run via a VMware hypervisor) with 8 VCPU with 12[31] GB RAM, named **devc3isp.iit.cnr.it**. It runs Ubuntu 16.04.3 LTS.

[29] When appropriate, matching requirements are reported between square brackets

[30] https://letsencrypt.org/ is a public benefit organisation that aims to spread the usage of encrypted web connections as much as possible, by releasing free X.509 certificates for Transport Layer Security (TLS)

[31] RAM has been improved with respect to 8GB planned in D7.1, because we observed higher memory requirements during source code compiling.

The base configuration includes the following software components:

- Oracle Java Development Kit, version 1.8 [C3ISP-Dev-008];
- GCC (GNU project C and C++ compiler), version 5.4 [C3ISP-Dev-009];
- Python, versions 2.7 and 3.5 [C3ISP-Dev-010].

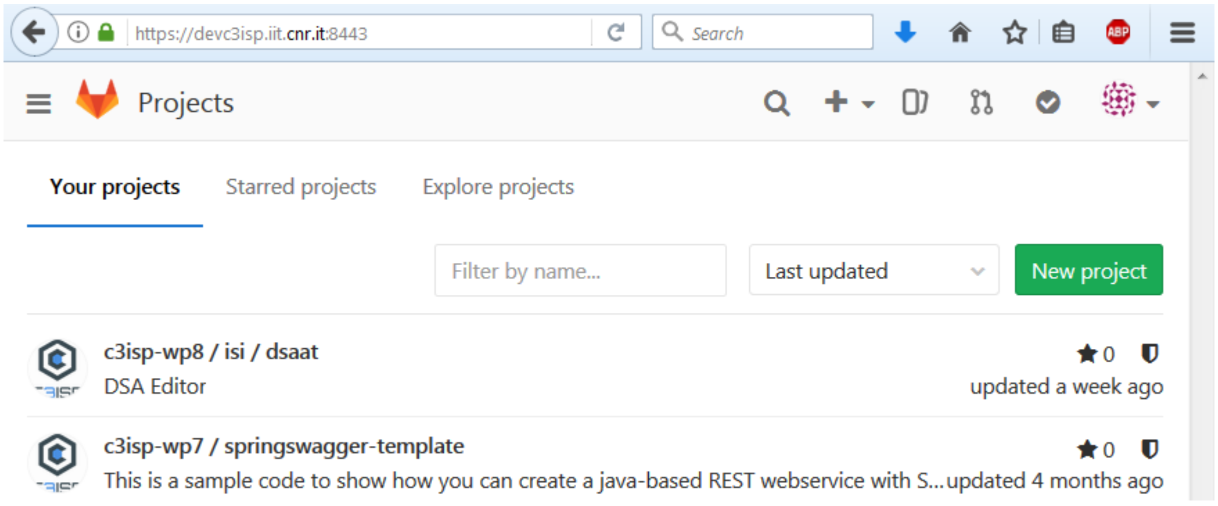
10.1.2. Software configuration

This section reports on the status of the development tools mentioned in deliverable D7.1. They support the process of continuous integration and delivery of software artefacts from the development to the test bed, as well as the evaluation of their quality and the assessment of their security posture. Our aim is to assure that the developed software is of high quality. Of course, these are "just" state-of-the-art tools, which need to be used in the context of a disciplined software development and engineering process.

The installed tools are the following:

- **GitLab Community Edition**[32] is a git-based distributed version control system that will host the C3ISP software repository. It addresses in particular requirement [C3ISP-Dev-001]. The configured service is available at https://devc3isp.iit.cnr.it:8443/.

  A sample screenshot is provided in the next figure:



c3isp-wp8 / isi / dsaat
DSA Editor
★ 0
updated a week ago

c3isp-wp7 / springswagger-template
This is a sample code to show how you can create a java-based REST webservice with S...
★ 0
updated 4 months ago

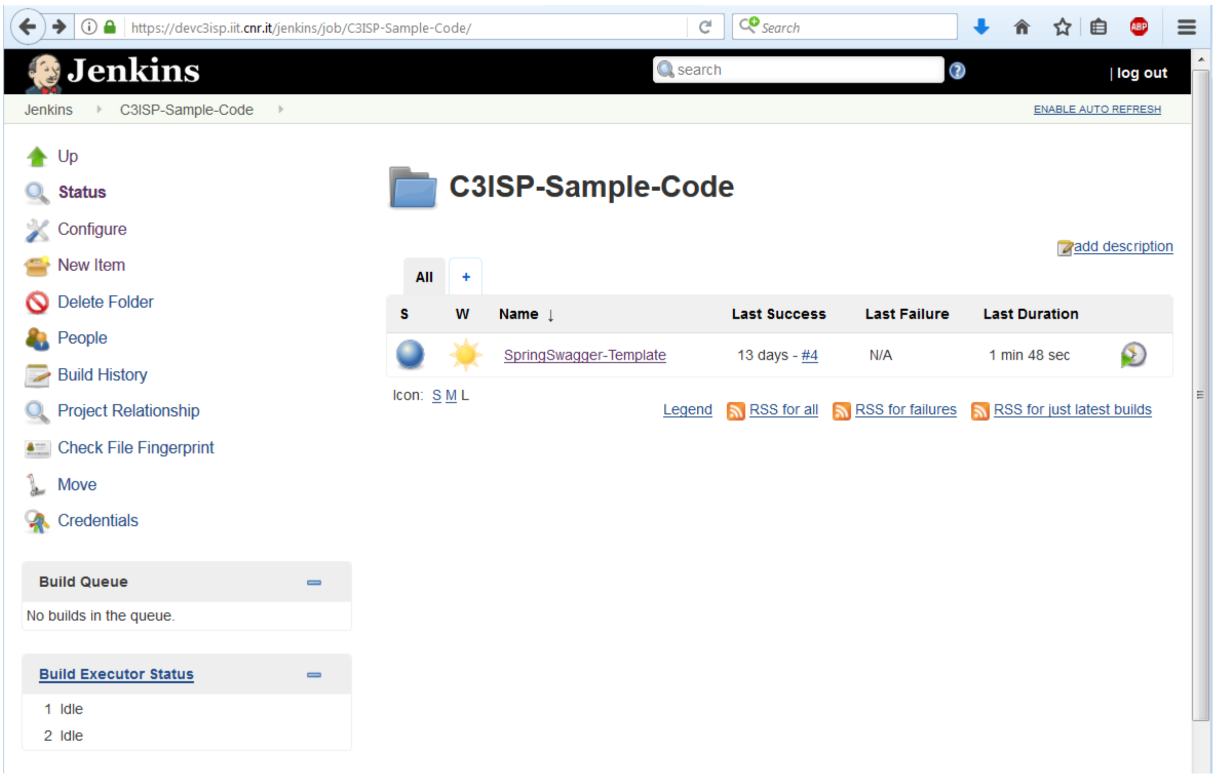
Figure 33: GitLab Web Interface

- **Jenkins**[33] is the tool which manages the continuous integration and deployment automation processes. It addresses in particular requirement [C3ISP-Dev-002]. The configured service is available at https://devc3isp.iit.cnr.it/jenkins. Most of the tools described next that are used to check the quality and security of the software artefacts are directly integrated with it.

[32] https://gitlab.com/gitlab-org/gitlab-ce

[33] https://jenkins.io/

A sample screenshot is provided in the next figure:


https://devc3isp.iit.cnr.it/jenkins/job/C3ISP-Sample-Code/

Jenkins › C3ISP-Sample-Code › ENABLE AUTO REFRESH

Up
Status
Configure
New Item
Delete Folder
People
Build History
Project Relationship
Check File Fingerprint
Move
Credentials

Build Queue
No builds in the queue.

Build Executor Status
1 Idle
2 Idle

C3ISP-Sample-Code

add description

S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☀	SpringSwagger-Template	13 days - #4	N/A	1 min 48 sec

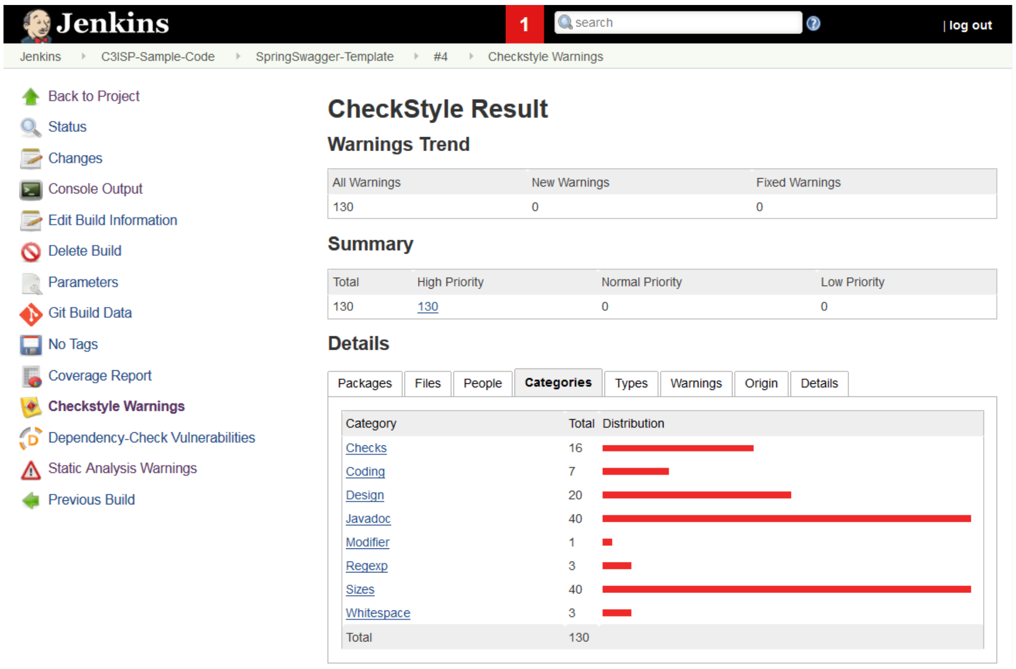
Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

Figure 34: Jenkins Web Interface

Jenkins has been currently integrated with some of quality and assurance tools described in deliverable D7.1, in particular:

- **CkeckStyle** for Java code syntactical checking and standard adherence. The next figure shows a sample of its output that highlights the categories of syntactical defects found:


Jenkins › C3ISP-Sample-Code › SpringSwagger-Template › #4 › Checkstyle Warnings

Back to Project
Status
Changes
Console Output
Edit Build Information
Delete Build
Parameters
Git Build Data
No Tags
Coverage Report
Checkstyle Warnings
Dependency-Check Vulnerabilities
Static Analysis Warnings
Previous Build

CheckStyle Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
130	0	0

Summary

Total	High Priority	Normal Priority	Low Priority
130	130	0	0

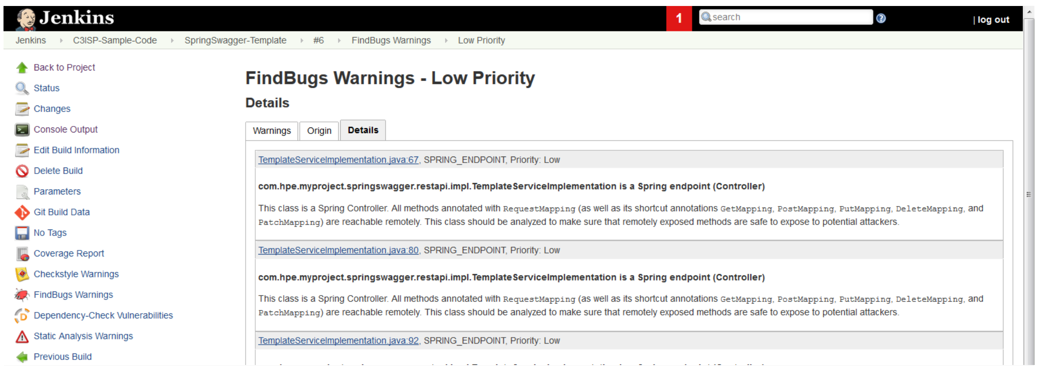
Details

Packages Files People Categories Types Warnings Origin Details

Category	Total	Distribution
Checks	16	
Coding	7	
Design	20	
Javadoc	40	
Modifier	1	
Regexp	3	
Sizes	40	
Whitespace	3	
Total	130	

Figure 35: CheckStyle Result Sample

- **FindBugs**[34] and **FindSecurityBugs** for Java code bugs discovering and security static analysis. The figure below illustrates few code issues found by these tools:



FindBugs Warnings - Low Priority

Details

TemplateServiceImplementation.java:67, SPRING_ENDPOINT, Priority: Low

com.hpe.myproject.springswagger.restapi.impl.TemplateServiceImplementation is a Spring endpoint (Controller)

This class is a Spring Controller. All methods annotated with `RequestMapping` (as well as its shortcut annotations `GetMapping`, `PostMapping`, `PutMapping`, `DeleteMapping`, and `PatchMapping`) are reachable remotely. This class should be analyzed to make sure that remotely exposed methods are safe to expose to potential attackers.

TemplateServiceImplementation.java:80, SPRING_ENDPOINT, Priority: Low

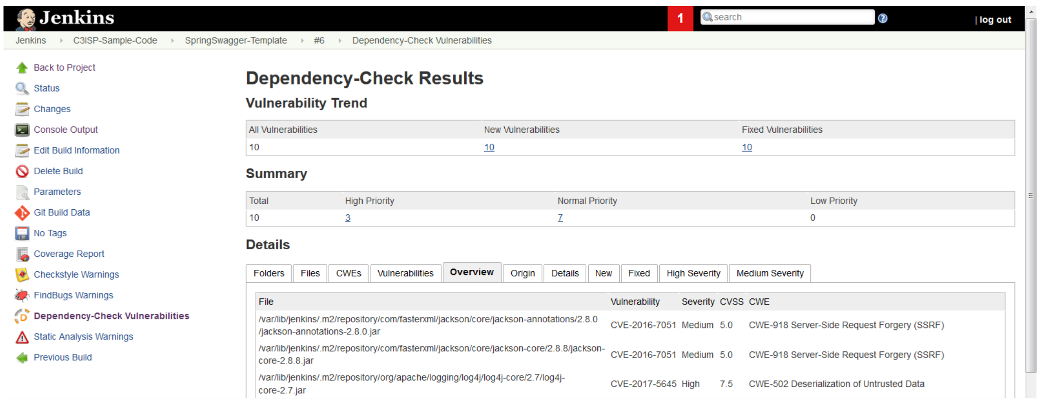
com.hpe.myproject.springswagger.restapi.impl.TemplateServiceImplementation is a Spring endpoint (Controller)

This class is a Spring Controller. All methods annotated with `RequestMapping` (as well as its shortcut annotations `GetMapping`, `PostMapping`, `PutMapping`, `DeleteMapping`, and `PatchMapping`) are reachable remotely. This class should be analyzed to make sure that remotely exposed methods are safe to expose to potential attackers.

TemplateServiceImplementation.java:92, SPRING_ENDPOINT, Priority: Low

Figure 36: FindBugs Result Sample

- **OWASP Dependency Check** for checking security vulnerabilities in external code dependencies (e.g. used Java libraries). The next figure shows some vulnerable libraries used in a sample code:



Dependency-Check Results

Vulnerability Trend

All Vulnerabilities	New Vulnerabilities	Fixed Vulnerabilities
10	10	10

Summary

Total	High Priority	Normal Priority	Low Priority
10	3	7	0

Details

File	Vulnerability	Severity	CVSS	CWE
/var/lib/jenkins/.m2/repository/com/fasterxml/jackson/core/jackson-annotations/2.8.0 /jackson-annotations-2.8.0.jar	CVE-2016-7051	Medium	5.0	CWE-918 Server-Side Request Forgery (SSRF)
/var/lib/jenkins/.m2/repository/com/fasterxml/jackson/core/jackson-core/2.8.8/jackson-core-2.8.8.jar	CVE-2016-7051	Medium	5.0	CWE-918 Server-Side Request Forgery (SSRF)
/var/lib/jenkins/.m2/repository/org/apache/logging/log4j/log4j-core/2.7/log4j-core-2.7.jar	CVE-2017-5645	High	7.5	CWE-502 Deserialization of Untrusted Data

Figure 37: OWASP Dependency Check Summary Sample

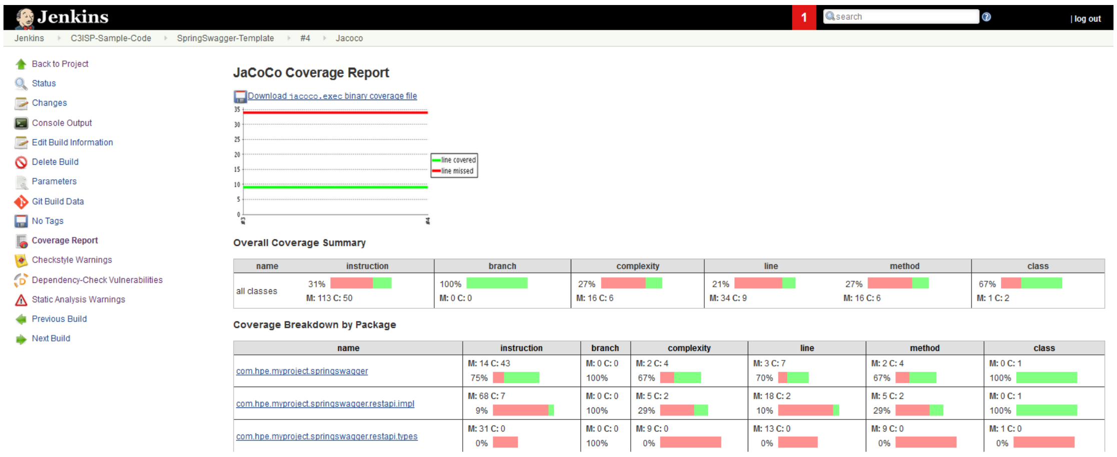
- We also integrated a tool to verify the coverage of the unit tests (the unit tests are created with the **Junit**[35] framework) with respect to the source code, which was not originally planned in D7.1. However, we think it is interesting for the overall project quality to be able to understand such coverage degree. We selected for this task the most well-known tool called **Cobertura**[36]. Cobertura measures the percentage of code covered by the unit test and allows the identification of part of code not involved in the tests. It can be easily integrated in a continuous integration environment, since it can be executed using Maven for analysing Java source code.

  The next figure shows a screenshot of it:

[34] FindBugs project has been superseded and is being migrated to its successor SpotBugs, https://spotbugs.github.io/. We plan to upgrade to SpotBugs once it will be stable and production ready.

[35] http://junit.org

[36] http://cobertura.github.io/cobertura/



JaCoCo Coverage Report

Download jacoco.exec binary coverage file

Overall Coverage Summary

name	instruction	branch	complexity	line	method	class
all classes	31% M: 113 C: 50	100% M: 0 C: 0	27% M: 16 C: 6	21% M: 34 C: 9	27% M: 16 C: 6	67% M: 1 C: 2

Coverage Breakdown by Package

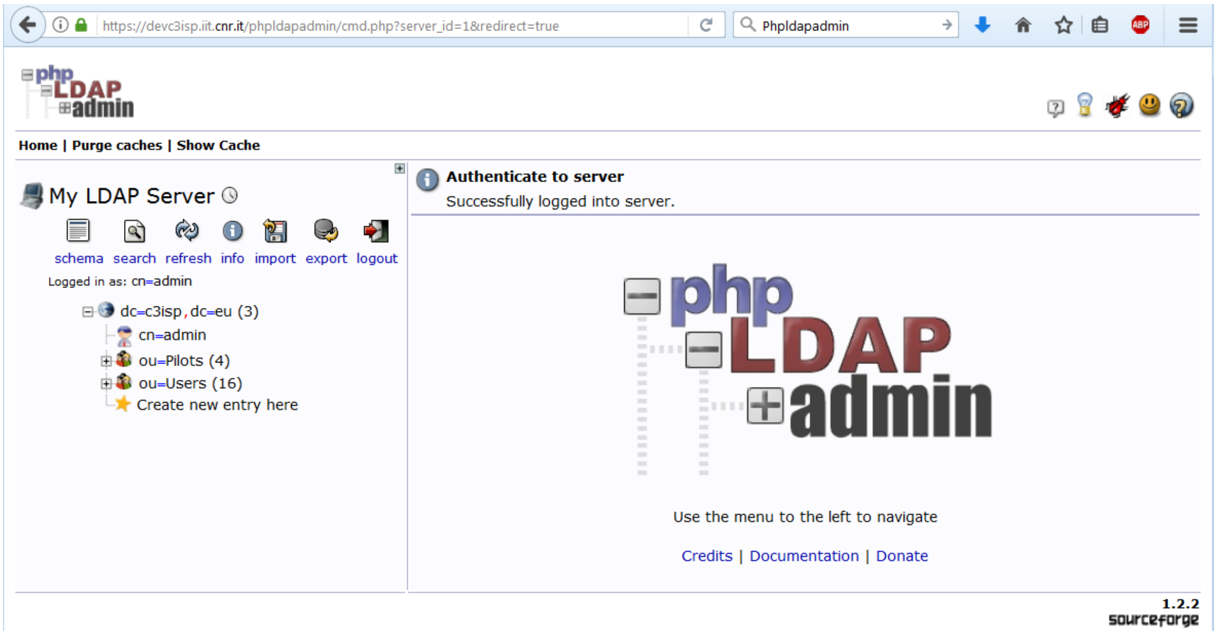
name	instruction	branch	complexity	line	method	class
com.hpe.myproject.springswagger	M: 14 C: 43 75%	M: 0 C: 0 100%	M: 2 C: 4 67%	M: 3 C: 7 70%	M: 2 C: 4 67%	M: 0 C: 1 100%
com.hpe.myproject.springswagger.restapi.impl	M: 68 C: 7 9%	M: 0 C: 0 100%	M: 5 C: 2 29%	M: 18 C: 2 10%	M: 5 C: 2 29%	M: 0 C: 1 100%
com.hpe.myproject.springswagger.restapi.types	M: 31 C: 0 0%	M: 0 C: 0 100%	M: 9 C: 0 0%	M: 13 C: 0 0%	M: 9 C: 0 0%	M: 1 C: 0 0%

Figure 38: JaCoCo Coverage Report Example

These tools have been verified with some custom Java sample code.

- **OpenLDAP**[37] is used as the common authentication Lightweight Directory Access Protocol (LDAP) repository for users signing into the development environment services. It has been integrated with all the services that require user authentication (e.g., GitLab and Jenkins), such in a way that user management is now centralised in OpenLDAP. We also installed **phpLDAPadmin** as a graphical front-end for managing users.

  The next picture provides a screenshot of the installed phpLDAPadmin:



Logged in as: cn=admin

Authenticate to server
Successfully logged into server.

Use the menu to the left to navigate

Figure 39: phpLDAPadmin Web Interface

- **Maven** and **make** tools have also been installed for build automation for Java, and C/C++.

The development environment is not yet completed. As reported in deliverable D7.1, we will shortly install the missing components, namely **Nexus** (the artefacts repository) and **Trac** (the

[37] https://www.openldap.org/

bug tracking system). We also have in roadmap the integration of the tools for C/C++ and python quality/security checkers (**CppCheck** and **Tox**), as well as the tool for dynamic code security checking (**OWASP Zap**).

The general schema of the integrated tools for the development environment is as follows:

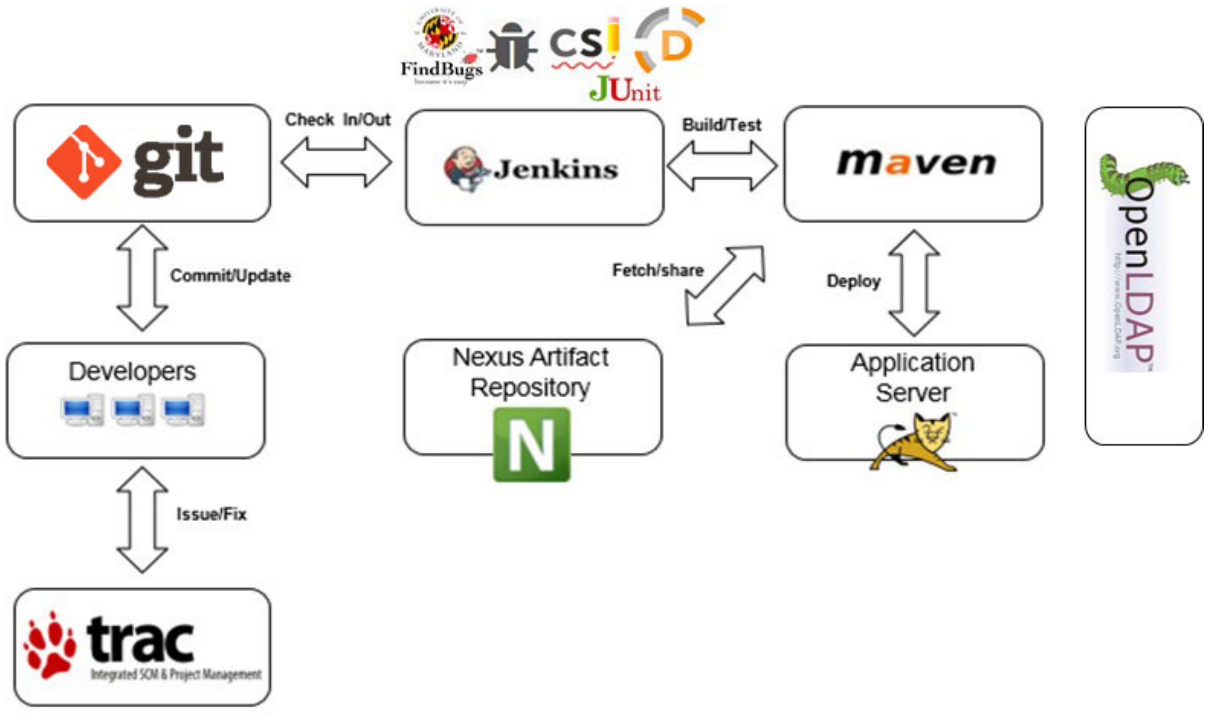


Figure 40: Development Environments Tools

The typical usage scenario is as follows:

1. The source code is pushed from the developers' workstations to the versioning system (git);
2. Automatically (or on demand), the build automation server (Jenkins) runs the continuous integration job that orchestrates the whole build process;
3. Jenkins uses Maven for Java (or Make for C/C++) to:
    a. Build the binaries artefacts from source code;
    b. Run the unit tests;
    c. Verify the test coverage;
    d. Evaluate the quality and security of the code (by invoking CheckSytle, FindBugs/FindSecurityBugs, OWASP Dependency Check);
    e. Deploy the resulting artefact to an application server (Tomcat) which is running on the Test Bed environment (thus linking the two environments);
4. Resulting artefacts are stored onto the artefacts repository (Nexus), e.g. to be shared between jobs (in case of libraries) or as historical archive;
5. Optionally, defects are filled into Trac by developers (it is worth noticing that Trac defects, git versions and Jenkins build numbers can be linked together to monitor issue resolution).

10.2. Test Bed Environment

The Test Bed Environment is at its very early stage since the development activities will start after M12 (which is why we concentrated so much on the preparation of the Development Environment, as described earlier). The following table maps the C3ISP subsystems to the Test Bed Environment:

Table 9 – C3ISP subsystems on Test Bed Environment

Subsystem	Virtual Machine	Notes
DSA Manager	isic3isp.iit.cnr.it	We opted to have these two subsystems on the same VM because of the small footprint the DSA Manager should have. Since all the components will be based on the micro-services paradigm, it will be easy to move to dedicated VM if necessary.
ISI	isic3isp.iit.cnr.it	
IAI	iaic3isp.iit.cnr.it	This VM has still to be created.

The rest of this section provides a short summary of the Test Bed Environment status.

10.2.1. Base configuration

In deliverable D7.1, we designed the Test Bed Environment and planned to have 4 VMs for the Pilots (one for each Pilot), a VM for the ISI and a VM for IAI.

As of now, we have installed the VM for ISI. It is based on a 2 VCPU (on VMware hypervisor) with 6 GB RAM, named **isic3isp.iit.cnr.it**. It runs Ubuntu 16.04.3 LTS. The Continuous Integration server (Jenkins) on the Development Environment feeds this machine by installing the built artefacts on it. We will setup shortly also the IAI VM, so that this environment will become finally the C3ISP reference implementation of the C3ISP framework architecture.

10.2.2. Software configuration

The currently installed package is only an application server where the Development Environment deploys the compiled artefacts:

- **Tomcat** v8.0.x, reachable at https://isic3isp.iit.cnr.it:8443/

11. Conclusions

This deliverable describes the first version (at M12) of the C3ISP Framework reference architecture, which is based on the requirements elicited in deliverable D7.1, as derived by the Pilots use cases. The architecture has been designed to address the Pilots requirements. WP7 had always (and continue to have) a close interaction with WP6 with checkpoints for tuning and steering the design effort to better fulfil the Pilots' needs. We also traced the design decisions with respect to the requirements (Section 9) in order to monitor and address the gaps. This deliverable describes each component and its interaction with the whole framework, and also defines different deployment models to address varying trust assumptions. It also identifies and explores the functions and operations the C3ISP Framework provides to both the Prosumers and Pilots' applications. The resulting C3ISP Framework reference architecture provides a generic and extensible framework that will serve as the basis for the next development activities.

The architecture will be refined as the integration of the Pilots into the C3ISP Framework progresses (WP2-5) and the core technologies matures (data sharing, analytics and crypto technologies in WP8). In fact, we plan that both WPs will continuously feed WP7 with inputs for design refinement during the whole project lifetime. In particular, the next M24 milestone will see the release of the first version of the C3ISP Framework that will take also into account this refinement process.

12. Appendix 1: Differential Privacy

Differential Privacy [75] is a privacy definition that aims to protect a single individual from harm that (s)he might suffer from participating in data collection while still allowing useful statistical analysis about a population (i.e. a group of individuals).

Differential Privacy requires knowledge about the entire value domain, e.g. what is the largest impact a record's inclusion/exclusion can have on a function evaluation. This impact is called *sensitivity* and is one of the two parameters of Differential Privacy. The other parameter, called *privacy parameter* or simply ϵ, quantifies the potential loss of privacy (or, from another point of view, the increase in accuracy) of the anonymisation.

Roughly, Differential Privacy works by adding noise scaled to the sensitivity of a function evaluation and a desired privacy parameter, i.e. it "hides" the involvement of a single individual while maintaining statistical inference over many individuals. Different ways to sample the noise or perturb the data (referred to as *mechanisms* in the literature) might be required (or need be developed) depending on what the purpose of the anonymised data is, i.e. what the desired function evaluation is. For example, for single geo-location points *Geo-Indistinguishability* was developed [72].

There are two models in Differential Privacy: the *interactive* and *non-interactive* model. In the former model one adds a noise scalar n to a (aggregation) function $f(\cdot)$ for a value vector V: $f(V) + n$; where n, as mentioned before, is parameterised to "hide" the maximum impact a record (i.e. one individual) can have on $f(\cdot)$. In the later model we add a noise vector N to the values V and afterwards perform the function evaluation: $f(V + N)$. The anonymisation tool that will be extended for C3ISP only supports the non-interactive model, i.e. the parameterised addition of noise to individual values. However, one could, for certain instances, call the tool on $f(V)$ instead of V and thus simulate the interactive model when properly parameterised.

As noted in [71] "*the Fundamental Law of Information Recovery states that overly accurate answers to too many questions will destroy privacy in a spectacular way*" and that this "*applies to all techniques for privacy-preserving data analysis, and not just to differential privacy.*" Thus, the number of (overly accurate) answers needs limitation. This can be controlled with the Usage Control model (see Continuous Authorization Engine in 4.1.1 and Obligation Engine in 4.1.2) and is outside the scope of the anonymisation itself.

13. Appendix 2: Homomorphic Computation

Homomorphic Computation enables an untrusted server to evaluate arithmetic circuits on ciphertexts without being able to decrypt inputs and outputs. In concrete terms, it is used to evaluate polynomials over encrypted bits.

Let us give a first example. Let us denote HE, a homomorphic encryption function. Consider a polynomial P(X,Y)=X+Y over integers and two integers 3 and 5. On clear data, the evaluation of the polynomial returns 8. In homomorphic cryptography, we manipulate encrypted data. In this case, we have two encryptions HE(3) and HE(5) of the two integers. The result of the evaluation of P over these ciphertexts is HE(8). That is an encryption of the expected result.

These polynomials can be multivariate and are described with **Boolean circuits**. An adapted way to describe a circuit is to use two Booleans gates: AND gates (binary multiplication) and XOR (binary addition). Two important parameters have to be minimized for practicability, the number of AND gates and the circuit **multiplicative depth**, that is the maximal number of AND gates between an input and an output of the circuit.

Let us consider a second example. Consider the polynomial Q(X,Y)=X^2*Y^2. We can evaluate this polynomial in different ways depending on operation order. A circuit permits to indicate the computation order. Let us take a first circuit representing how Q(X,Y) is computed:

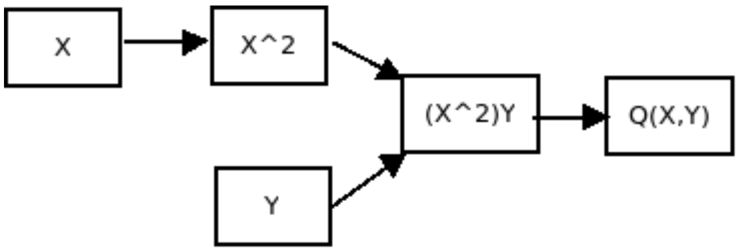


Figure 41: A Boolean circuit – un-optimised

The left boxes represent the circuit inputs. The right box is the circuit output. With this representation, the multiplicative depth is the maximal number of arrows between an input and an output of the circuit. Here, it is 3. We can do better (that is minimising the multiplicative depth) by changing the order of computations:

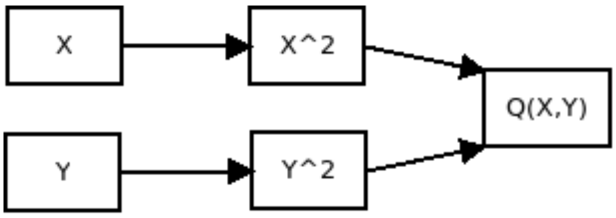


Figure 42: A Boolean circuit – optimised

In this manner, the multiplicative depth is minimized. It is only 2. This permits to decrease time and memory requirements.

Ideally, the multiplicative depth should be less than 20. Time and memory needs mainly depend on security level and multiplicative depth. There is no standardised homomorphic cryptosystem. This branch of cryptography began in 2009 [37]. Earlier, most of the homomorphic cryptosystem proposed only one homomorphic operation (addition or multiplication for instance) and thus less applications were proposed. Note that a homomorphic scheme is probabilistic, so an attacker which has only access to ciphertexts could decrypt it (non-specialists can refer to [88] for more information).

<u>Specific requirements for HE computation using Cingulata tool</u>

Cingulata tool needs constant data length as input, which requires a precomputation for certain data types. Indeed, IPv4 addresses satisfy this requirement because they are represented by 4 bytes, while strings do not (string size is a parameter to consider).

Strings representations depend on:

- Character encoding;
- String size.

In our implementation, character encoding is (extended) ASCII, where a character is represented with one byte. To address the non-constant string size issue, a solution is to encode the string with zero padding and truncation. The encoded data is then stored as a fixed number of bytes (this number is an additional parameter to consider).

Under those assumptions, the data representation becomes:

- Each IPv4 is stored as 4 bytes;
- Each ASCII character is stored as 1 byte;
- Each ASCII string is stored as X bytes, where X is a parameter;

We choose IPv4 rather than IPv6 addresses and ASCII rather than UTF-8 encoding, because they both differ in data representation size (they both use more bytes): in homomorphic cryptography, this parameter can have a significant impact on time and memory requirements.

To sum up, the parameters in our FHE analytics are:

- The input data (IPv4s, strings);
- Encoded data size (a constant integer for IPv4s, an integer parameter for strings);
- Number of data (the list size is a parameter).

14. References

This section lists the references used throughout the document:

[1] K. Brennan, A Guide to the Business Analysis Body of Knowledge, International Institute of Business Analysis, 2009.

[2] STIX™ – Structured Threat Information Expression, https://oasis-open.github.io/cti-documentation/, https://stixproject.github.io/, fetched on March 16th, 2017

[3] Guide to Cyber Threat Information Sharing, NIST Special Publication 800-150, http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-150.pdf, fetched on March 16th, 2017

[4] CEF – Common Event Format, https://www.protect724.hpe.com/docs/DOC-1072, fetched on March 16th, 2017

[5] B. Kepes, Cloudonomics: the Economics of Cloud Computing, Rackspace Hosting, Diversity Limited, Aug. 2011

[6] Gartner, Gartner Says By 2020, a Corporate "No-Cloud" Policy Will Be as Rare as a "No-Internet" Policy Is Today, http://www.gartner.com/newsroom/id/3354117, Jun. 2016, fetched on March 16th, 2017

[7] CybOX™ – Cyber Observable eXpression, https://oasis-open.github.io/cti-documentation/, https://cyboxproject.github.io/, fetched on March 16th, 2017

[8] TAXII™ – Trusted Automated eXchange of Indicator Information, https://oasis-open.github.io/cti-documentation/, https://taxiiproject.github.io/, fetched on March 16th, 2017

[9] OpenC2 – Open Command and Control, http://openc2.org/, fetched on March 16th, 2017

[10] S, Carpov; T.H. Nguyen; R. Sirdey; G. Constantino; F. Martinelli; Practical Privacy-Preserving Medical Diagnosis Using Homomorphic Encryption

[11] S. Carpov, P. Dubrulle, R. Sirdey, "Armadillo: a compilation chain for privacy preserving applications", Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (3rd International Workshop on Security in Cloud Computing), pp. 13-19, 2015

[12] Borghoff J. et al. (2012) PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications. In: Wang X., Sako K. (eds) Advances in Cryptology – ASIACRYPT 2012. ASIACRYPT 2012. Lecture Notes in Computer Science, vol 7658. Springer, Berlin, Heidelberg

[13] Albrecht M.R., Rechberger C., Schneider T., Tiessen T., Zohner M. (2015) Ciphers for MPC and FHE. In: Oswald E., Fischlin M. (eds) Advances in Cryptology -- EUROCRYPT 2015. EUROCRYPT 2015. Lecture Notes in Computer Science, vol 9056. Springer, Berlin, Heidelberg

[14] Canteaut A. et al. (2016) Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. In: Peyrin T. (eds) Fast Software Encryption. FSE 2016. Lecture Notes in Computer Science, vol 9783. Springer, Berlin, Heidelberg

[15] ECRYPT - European Network of Excellence in Cryptology: The eSTREAM StreamCipher Project (2005).

[16] De Cannière C., Preneel B. (2008) Trivium. In: Robshaw M., Billet O. (eds) New Stream Cipher Designs. Lecture Notes in Computer Science, vol 4986. Springer, Berlin, Heidelberg

[17] N. Bouzerna, R. Sirdey, O. Stan, T.-H. Nguyen and P. Wolf, "An architecture for practical confidentiality-strengthened face authentication embedding homomorphic cryptography", Proceedings of the 8th IEEE International Conference on Cloud Computing Technology and Science, pp. 399-406, 2016.

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

[19] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An Industry-Standard API for Shared-Memory Programming. IEEE Comput. Sci. Eng. 5, 1 (January 1998), 46-55. DOI=http://dx.doi.org/10.1109/99.660313

[20] Berkeley Verification and Synthesis Research Center, A System for Sequential Synthesis and Verification https://bitbucket.org/alanmi/abc

[21] Stober, Thomas –Hansmann, Uwe "Agile Software Development: Best Practices for Large Software Development Projects" -Springer 2009

[22] Bass, Len Ingo Weber, Zhu Liming – Hansmann, Uwe DevOps: A Software Architect's Perspective" –Addison-Wesley Professional 2015

[23] A Python module for creating JUnit XML test result documents, https://pypi.python.org/pypi/junit-xml, fetched on March 16th, 2017

[24] NIST Glossary of Key Information Security Terms, http://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7298r2.pdf, fetched on March 16th, 2017

[25] Gartner Newsroom, http://www.gartner.com/newsroom/id/3354117, fetched on March 16th, 2017

[26] I. Matteucci, M. Petrocchi, and M. L. Sbodio. CNL4DSA: a Controlled Natural Language for Data Sharing Agreements. In SAC: Privacy on the Web Track. ACM, 2010. 21, 23, 52

[27] Grigoris Antoniou and Frank Van Harmelen. Web Ontology Language: OWL. In Handbook on Ontologies in Information Systems, pages 67–92. Springer, 2003. 18

[28] Definition of Document Oriented Database https://www.techopedia.com/definition/30329/document-oriented-database

[29] Original BSD syslog, https://tools.ietf.org/html/rfc3164

[30] RFC5424, https://tools.ietf.org/html/rfc5424

[31] "Common Event Format" , ArcSight, Inc. July 22, 2010 Revision 16, https://kc.mcafee.com/resources/sites/MCAFEE/content/live/CORP_KNOWLEDGEBASE/78000/KB78712/en_US/CEF_White_Paper_20100722.pdf

[32] Representational State Transfer (REST) http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[33] Unified Modeling Language - UML, http://www.uml.org/

[34] Pub, NIST FIPS. "197: Advanced encryption standard (AES)." Federal information processing standards publication 197.441 (2001): 0311.

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

[36] Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC press.

[37] Gentry, Craig. "Fully homomorphic encryption using ideal lattices." *STOC*. Vol. 9. No. 2009. 2009.

[38] Barker, Elaine, et al. "Recommendation for key management part 1: General (revision 3)." *NIST special publication* 800.57 (2012): 1-147. Key Management Guidelines SP 800-57 Part 2, Best Practices for Key Management Organizations

[39] Key Management Guidelines SP 800-57 Part 3, Application-Specific Key Management Guidance

[40] Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM)

[41] Shirey, R. "RFC 2828–Internet security glossary, 2000." *URL: http://www.faqs.org/rfcs/rfc2828. Html* (2003).

[42] Oppliger, Rolf. *Contemporary cryptography*. Artech House, 2011.

[43] RFC 6239, Suite B Cryptographic Suites for Secure Shell (SSH)

[44] RFC 6379, Suite B Cryptographic Suites for Ipsec

[45] RFC 6460, Suite B Profile for Transport Layer Security (TLS)

[46] Lyubashevsky, Vadim, Chris Peikert, and Oded Regev. "On ideal lattices and learning with errors over rings." *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Berlin, Heidelberg, 2010.

[47] Fontaine, Caroline, and Fabien Galand. "A survey of homomorphic encryption for non specialists." *EURASIP Journal on Information Security* 2007.1 (2007): 013801.

[48] Gentry, C., S. Halevi, and N. P. Smart. *Homomorphic evaluation of the AES circuit (updated implementation)*. IACR Cryptology ePrint Archive: Report 2012/099, https://eprint. iacr. org/2012/099. Pdf, 2015.

[49] *FIPS PUB 140-2" (PDF). NIST. 2002-12-03. Retrieved 2017-03-31.*http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf

[50] Canteaut, Anne, et al. "Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression." *International Conference on Fast Software Encryption*. Springer Berlin Heidelberg, 2016.

[51] De Canniere, Christophe, and Bart Preneel. "Trivium." *New Stream Cipher Designs* (2008): 244-266.

[52] Amazon S3 Data Lake https://aws.amazon.com/big-data/data-lake-on-aws/

[53] JavaScript Object Notation official site: http://json.org/

[54] Hadoop Distributed File System (HDFS) URL: https://hadoop.apache.org/docs/r1.2.1/hdfs_user_guide.html

[55] OpenStack™ Swift URL: https://docs.openstack.org/swift/latest/

[56] MongoDB official web site: https://www.mongodb.com/

[57] RFC 6749, OAuth2: https://tools.ietf.org/html/rfc6749, http://oauth.net/2/

[58] Unity 3D: https://unity3d.com/

[59] A One-Time Password System: https://www.rfc-editor.org/rfc/rfc2289.txt

[60] Lightweight Directory Access Protocol: https://msdn.microsoft.com/en-us/library/aa367008(v=vs.85).aspx

[61] OpenID Specification: http://openid.net/developers/specs/

[62] Apache Hive, URL: https://hive.apache.org/

[63] Cloudera Impala, URL: https://www.cloudera.com/products/open-source/apache-hadoop/impala.html

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

[65] Oppliger, Rolf. Contemporary cryptography. Artech House, 2011

[66] Fundamental modeling concepts (FMC), URL: http://www.fmc-modeling.org

[67] Sticky Policy: An Approach for Managing Privacy across Multiple Parties, URL: http://ieeexplore.ieee.org/document/5959137/

[68] PostgreSQL Database, URL: https://www.postgresql.org/

[69] Transport Layer Security, URL: https://tools.ietf.org/html/rfc5246

[70] LDAP over SSL, URL: https://social.technet.microsoft.com/wiki/contents/articles/2980.ldap-over-ssl-ldaps-certificate.aspx

[71] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. Foundations and Trends in Theoretical Computer Science, 2014.

[72] Miguel E Andrés, Nicolás E Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. *Geo-indistinguishability: Differential privacy for location-based systems*. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013

[73] MySQL Database, URL: https://www.mysql.com/it/

[74] Oracle DBMS, URL: https://www.oracle.com/database/index.html

[75] Roth, Cynthia Dwork and Aaron, The algorithmic foundations of differential privacy, Foundations and Trends in Theoretical Computer Science, 2014