CyberKit4SME
Democratizing a Cyber Security Toolkit for SMEs and MEs
Project Nº 883188

D5.1
Initial Services for Data Protection

Responsible: Maya Anderson, IBM
Contributors: Maya Anderson, IBM
Stefano De Angelis, University of Southampton - Cyber

Document Reference: D5.1
Dissemination Level: Public
Version: 1.0
Date: 2021-09-29

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 883188.
Executive Summary

This report on the initial services for data protection summarizes the work done and tools provided for the testbed (WP2) and for the SMEs (WP3) during the first project reporting period. It presents the Secure Data Services and the Service Ledger, describing their goals, architectures and interfaces.
## Contributors Table

<table>
<thead>
<tr>
<th>DOCUMENT SECTION</th>
<th>AUTHOR(S)</th>
<th>REVIEWER(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction, Conclusions, References</td>
<td>Maya Anderson (IBM), Stefano De Angelis (UoS Cyber)</td>
<td>Tiberiu Cocias (ElektroBit), Aurélien Lecamus (Inetum)</td>
</tr>
<tr>
<td>Secure Data Services and Run-time isolation and protection facilities</td>
<td>Maya Anderson (IBM)</td>
<td>Tiberiu Cocias (ElektroBit), Aurélien Lecamus (Inetum)</td>
</tr>
<tr>
<td>Security information sharing and collaboration framework</td>
<td>Stefano De Angelis (IBM)</td>
<td>Tiberiu Cocias (ElektroBit), Aurélien Lecamus (Inetum)</td>
</tr>
</tbody>
</table>
# Table of Contents

## I. INTRODUCTION
- Purpose and organization of the document ................................................................. 8
- Scope and audience .......................................................................................................... 8

## II. SECURE DATA SERVICES AND RUN-TIME ISOLATION AND PROTECTION FACILITIES
- Background ..................................................................................................................... 9
- Goals .............................................................................................................................. 11
  - Secure Data Services Goals and Features .................................................................. 11
  - Run-time isolation and protection facilities Goals and Features ............................... 12
- Overview ....................................................................................................................... 13
- Architecture .................................................................................................................. 15
  - SDS with HTTP API ................................................................................................. 16
  - SDS deployed as a library ......................................................................................... 17
- Implementation Details ............................................................................................... 17
  - PyArrow with PME .................................................................................................. 17
  - Key Management Service ......................................................................................... 17
  - Configuration of PME ............................................................................................. 18
- Interfaces ....................................................................................................................... 20
  - SDS Interfaces with other Tools within the CK4SME Toolkit ................................. 20
    - SDS <> Keenaï ...................................................................................................... 20
  - SDS Interfaces for Use cases .................................................................................. 21
    - SDS HTTP API – SQL Query ............................................................................. 21
    - SDS HTTP API – Bulk import .......................................................................... 23
    - SDS HTTP API – Insert Data ............................................................................ 25
- Usage ............................................................................................................................. 26
- Validation ....................................................................................................................... 28
- Next Steps ..................................................................................................................... 29

## III. SECURITY INFORMATION SHARING AND COLLABORATION FRAMEWORK
- Background .................................................................................................................. 30
  - CTI Sharing: STIX and TAXII standards ................................................................. 30
  - The Blockchain Technology ..................................................................................... 31
  - Decentralised Data Storage ..................................................................................... 32
- Goals .............................................................................................................................. 33
- Overview ....................................................................................................................... 34
- Architecture ................................................................................................................ 36
- Implementation Details ............................................................................................. 38
- Interfaces ....................................................................................................................... 38
<table>
<thead>
<tr>
<th>Table of Figures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1 Encrypted Parquet file format</td>
</tr>
<tr>
<td>Figure 2 File tampering attack</td>
</tr>
<tr>
<td>Figure 3 SDS overview</td>
</tr>
<tr>
<td>Figure 4 SDS in the Public cloud</td>
</tr>
<tr>
<td>Figure 5 SDS in Hybrid cloud</td>
</tr>
<tr>
<td>Figure 6 SDS High-level architecture</td>
</tr>
<tr>
<td>Figure 7 SDS with HTTP API</td>
</tr>
<tr>
<td>Figure 8 SDS deployed as a library</td>
</tr>
<tr>
<td>Figure 9 Hashicorp Vault Key Management Service</td>
</tr>
<tr>
<td>Figure 10 Different keys for different columns</td>
</tr>
<tr>
<td>Figure 11 SDS HTTP API – OpenAPI live documentation</td>
</tr>
<tr>
<td>Figure 12 OpenAPI specification of SQL query</td>
</tr>
<tr>
<td>Figure 13 cURL SQL query</td>
</tr>
<tr>
<td>Figure 14 Example application with SQL query</td>
</tr>
<tr>
<td>Figure 15 OpenAPI specification of Bulk import</td>
</tr>
</tbody>
</table>
D5.1 Initial Services for Data Protection

Figure 16 cURL Bulk import ................................................................. 24
Figure 17 Example application with Bulk import ........................................... 24
Figure 17 OpenAPI specification of Insert data .............................................. 25
Figure 19 Example application with Insert data ............................................ 25
Figure 20 Example application with Insert data ............................................ 26
Figure 20 SDS with HTTP API in a Financial use case ..................................... 26
Figure 22 SDS as a library in a Transportation use case .................................. 27
Figure 23 SDS Roadmap ............................................................................ 29
Figure 24 TAXII Collection and Channel ...................................................... 31
Figure 25 – Blockchain data structure .......................................................... 32
Figure 26. Overview of the Service Ledger Instance in CyberKit4SME ............... 34
Figure 27. High level overview of the SL CTI sharing platform architecture ....... 36
Figure 28 Detailed architecture of a SL instance .......................................... 37
Figure 29 – Structure of SL TAXII Server endpoints .................................... 39
Figure 30 SL Roadmap ............................................................................. 52

List of Tables

Table 1 - PME properties ............................................................................. 18
Table 2 – SL Discovery endpoints ............................................................... 39
Table 3 - SL Discovery API ........................................................................ 39
Table 4 – Discovery resource type ............................................................... 40
Table 5 – Error resource type .................................................................... 40
Table 6 – SL API Root API ........................................................................ 41
Table 7 – API Root resource type ............................................................... 41
Table 8 – SL Collection endpoints .............................................................. 42
Table 9 – SL Collections API ..................................................................... 42
Table 10 – Collection resource type ............................................................ 43
Table 11 – SL GET Objects API ................................................................. 44
Table 12 – Envelope resource type ............................................................. 45
Table 13 - SL ADD Objects API ................................................................. 45
Table 14 – SL Status API ......................................................................... 46
Table 15 – Status resource type ................................................................. 47
Table 16 – Status-details resource type ......................................................... 47
Table 17 – SL GET Object API ................................................................. 48
Table 18 – SL GET Object Versions API ..................................................... 49
Table 19 – Versions resource type ............................................................. 49
## Table of Acronyms and Definitions

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAD</td>
<td>Additional Authenticated Data</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CERT</td>
<td>Community Emergency Response Team</td>
</tr>
<tr>
<td>CID</td>
<td>Content Identifier</td>
</tr>
<tr>
<td>CSIRT</td>
<td>Cyber Security Incident Response Team</td>
</tr>
<tr>
<td>CTI</td>
<td>Cyber Threat Intelligence</td>
</tr>
<tr>
<td>DDS</td>
<td>Decentralised Data Storage</td>
</tr>
<tr>
<td>DR</td>
<td>Decentralised Repository</td>
</tr>
<tr>
<td>ForEx</td>
<td>Foreign Exchange</td>
</tr>
<tr>
<td>GDPR</td>
<td>General Data Protection Regulation</td>
</tr>
<tr>
<td>IoC</td>
<td>Indicator of Compromise</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>IPFS</td>
<td>InterPlanetary File System</td>
</tr>
<tr>
<td>JWT</td>
<td>JSON Web Token</td>
</tr>
<tr>
<td>KMS</td>
<td>Key Management Service</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>ME</td>
<td>Medium-sized Enterprise</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-peer</td>
</tr>
<tr>
<td>PME</td>
<td>Parquet Modular Encryption</td>
</tr>
<tr>
<td>PPoS</td>
<td>Pure Proof of Stake</td>
</tr>
<tr>
<td>SDS</td>
<td>Secure Data Service</td>
</tr>
<tr>
<td>SIEM</td>
<td>Security Information and Event Management</td>
</tr>
<tr>
<td>SL</td>
<td>Service Ledger</td>
</tr>
<tr>
<td>SME</td>
<td>Small and Medium-sized Enterprise</td>
</tr>
<tr>
<td>SoTA</td>
<td>State of The Art</td>
</tr>
<tr>
<td>SCO</td>
<td>STIX Cyber-observable Object</td>
</tr>
<tr>
<td>SDO</td>
<td>STIX Domain Object</td>
</tr>
<tr>
<td>SRO</td>
<td>STIX Relationship Object</td>
</tr>
<tr>
<td>STIX</td>
<td>Structured Threat Information eXpression</td>
</tr>
<tr>
<td>SSM</td>
<td>System Security Modeller</td>
</tr>
<tr>
<td>TTP</td>
<td>Tactics techniques and Procedures</td>
</tr>
<tr>
<td>TAXII</td>
<td>Trusted Automated Exchange of Intelligence Information</td>
</tr>
<tr>
<td>UoS</td>
<td>University of Southampton</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

I.1. Purpose and organization of the document

This report on the initial services for data protection summarizes the work done and tools provided for the testbed (WP2) and for the SMEs (WP3) during the first project reporting period. The first part of the document describes Secure Data Services and Run-time isolation and protection facilities together, and the second part focuses on Security information sharing and collaboration framework. Each part describes the background, goals, overview, architecture, implementation details, interfaces, usage, validation and next steps. At the end there is a conclusion and references.

I.2. Scope and audience

D5.1 Initial Services for Data Protection is based on requirements and architecture defined in WP2, in particular in D2.1 Requirements analysis [1] and D2.2 Technical specification [2]. It aims to address the needs of the Use cases defined in the scope of WP3 validation scenarios. This deliverable presents details about the interactions between the Secure Data Service and other tools that are described in D4.1 Initial risk analysis, monitoring and forecasting tools [4].
II. SECURE DATA SERVICES AND RUN-TIME ISOLATION AND PROTECTION FACILITIES

II.1. Background

Apache Parquet [10], [11] is the industry-leading standard for the formatting, storage and efficient processing of big data. Its storage-agnostic compression and advanced in-storage filtering have gained it wide recognition, so it is supported by most analytics tools, and also by the newest table formats, such as Apache Hudi, Iceberg and Delta Lake [5]. IBM Research initiated and led joint work with the Apache Parquet community on Parquet Modular Encryption (PME) [6] to address critical issues in securing the confidentiality and integrity of sensitive data, without degrading the performance of analytic systems.

Parquet Modular Encryption protects sensitive data at rest from the standpoint of both data privacy (encryption) and of data integrity (tamper-proofing) and can work in any untrusted storage – public cloud, private cloud, hybrid cloud, file system, etc. It encrypts Parquet files module-by-module — the footer, page headers, column indexes, offset indexes, pages, etc. Thus, it not only enables granular control of the data based on access to per-column encryption keys, it also preserves all the benefits of efficient analytics on Parquet. This includes column projection and predicate push-down, where entire file parts can be skipped if the metadata indicates that the part has no matching values. PME can be used for granular access control to data with per-column encryption keys and for safe and efficient data sharing where there is no need to copy, extract and re-encrypt data for sharing with multiple users or roles, but only access to the master encryption keys can be controlled.

PME provides data privacy by hiding sensitive information from unauthorized parties with full encryption of all data and metadata parquet modules. It includes minimum/maximum values, data schema, encryption key IDs and the list of sensitive columns. PME provides the ability to use separate keys for sensitive columns, for example based on their sensitivity levels, where data and metadata of the same column are encrypted with the same key. The encryption keys and the plaintext data are not accessible to the storage server administrators because of the client-side encryption.

The following figure presents the encrypted parquet structure, where the footer is encrypted with a separate footer key [6]. The blue parts highlight the Parquet format parts needed by a Parquet reader in order to navigate the file.
In addition to protecting the confidentiality of data, PME allows to protect data integrity, which is important too in order to make sure that file content is not tampered with by modifying either the data or the metadata. For example, in Figure 2 an attacker replaces one of the data pages with another data page, taken from an older parquet file encrypted with correct encryption keys. Usage of the AES GCM cipher in PME protects against byte replacement inside a ciphertext and the described attack example would be detected. In addition, PME enables the client to make sure that a file is not replaced with a wrong file, such as an older version of the same file. For example, altering healthcare data where medical sensor readings go missing or are replaced with their older values might affect the medical diagnosis and the medical treatment plan. The usage of AAD (additional authenticated data) enables to detect the use of an incorrect version of the file. However, in order to extend the integrity protection from file-level protection to table protection, we have to consider what constitutes a table. A table is a dynamic collection of files, and it should be protected against file removal or insertion. To this end a user can either manage AADs externally to parquet, or a new table format can be extended to support PME with integrity protection, e.g. Hudi or Iceberg.
II.2. Goals

The main goals of the Secure Data Services and Run-time isolation and protection facilities as described in the DoA of CyberKit4SME [14] are:

```
"... easily consumable services ... to guarantee the security and integrity of data ...

".. create a ... service to store data and consume data in a secure ... manner"

"...deployed on the cloud or multi-cloud"

"...leverage the newly approved Parquet encryption standard"

"...deliver a simple user experience of SQL-based operations on data encrypted at-rest"

"...access control and governance"

"...data integrity and validation of the stored data"

"... create a framework ... to share security information from security monitoring features ... and the data security and integrity services"
```

II.2.1. Secure Data Services Goals and Features

The goal of the Secure Data Services is to create easily consumable services to protect data across its lifecycle as it is stored, accessed and used. This includes facilities to govern data access and validation across the whole data lifecycle. The services to be created will aim to include:

- Data encryption and key management tools and facilities
o Protection for data-at-rest to be provided by using encrypted data formats (e.g., Apache Parquet).

o Encryption keys will be maintained in an isolated Key Management System (KMS), such as Hashicorp Vault [7]. This can be provided as a service for simple configuration and deployment of a KMS which will work together with the data format SDK and libraries (e.g., Parquet encryption) and enable access to keys according to policies.

- Portable access and governance. Enable access management and enforcement of policies independent of the physical store. For example, explore and demonstrate how to define and enforce policies by leveraging encryption and demonstrate how to manage access control for enforcing governance.

- Integrity validation of data. We plan to design and build multi-layer tools for protection of data integrity at various levels, such as:
  
  o Object and file level integrity to protect against modification of object/file contents or the replacement of an original object/file with a fabricated or legitimate but outdated data.
  
  o Data sets/table integrity to protect against the removal of table partition files or addition of fabricated files to tables.
  
  o Dynamic data sets/table integrity: to protect against attacks on the table snapshot state when data is added from streams, and old data legitimately removed during runtime.
  
  o Data-in-transit integrity to protect individual messages from modification or replacement, or injection of fabricated messages or the removal of legitimate messages from streams.
  
  o Data-in-use integrity: to protect against tampering with in-memory data.

II.2.2. Run-time isolation and protection facilities Goals and Features

This task will build a run-time mechanism for supporting data processing and consumption of various data types, and provide building blocks for access, isolation, governance and audit. The mechanism will be built to run on top of open environments such as Kubernetes [15] (open-source system for automating deployment, scaling, and management of containerized applications) and OpenShift [16] (hybrid cloud, enterprise Kubernetes application platform), and will be based on a pluggable approach to allow it to be expanded to multiple use-cases and sectors.

The run-time mechanisms and features will include:

- Data access modules to simplify access to data, allowing SMEs to consume data more easily, and benefit from secure data services and sharing (in Task 5.1).

- Latency and performance optimization features allowing efficient data access in a manner agnostic to the application development effort.

- Audit and inspection of data and processing to validate business and regulatory requirements, which will integrate with the data sharing platform built in Task 5.3.
II.3. Overview

Secure Data Service (depicted in Figure 3) provides a simple data interface to use cases with such APIs as:

- SQL service, to insert/query/delete data
- Data export and bulk exchange
  - Use Parquet format for data export and bulk exchange instead of CSV, or nd-JSON both for security and efficiency
- Machine Learning API

Since the goal of the CyberKit4SME toolkit, which includes the SDS and is described in D2.2 of WP2 [2], is to help SMEs with securing their data, but without incurring the burden of employing security experts, the data service shall provide security as transparently as possible using Parquet encryption, which is a new industry standard for data security, developed by IBM Research with the Open Source community. Parquet key management tools are used for managing and accessing encryption keys stored in Key Management Service. In order for the use cases to be able to rely on the data, the Secure Data Service enables data integrity validation, based on both the Parquet encryption standard for file integrity protection and on new development of dataset integrity validation. Access control and governance are also an important part of secure access to data, so the SDS will use access control to keys in the Key Management Service in order to control access to data encrypted with the help of these encryption keys.

SDS can be deployed both on-prem and in the public or private Cloud, based on use case requirements. The following patterns are useful in the Hybrid cloud and Data lake scenarios:

- SME can move most of use case load (compute, storage) to the Cloud in order to cut maintenance costs and efforts, as shown in Figure 4
- SME can use the Cloud to store secure backups and efficient restore of data in order to protect against ransomware attacks.
Figure 4 SDS in the Public cloud

Hybrid Cloud allows saving a significant investment in building and maintaining an on-premises IT infrastructure, along with the expensive IT and security in-house specialists required to plan and deploy such systems. Much of this burden is offloaded to the public cloud that is fully staffed with IT and security experts, who provide efficient support to a large number of cloud users and therefore provide advanced but affordable services to the customers. The customers, such as the hospitals, can focus on their main area of professional expertise - and also can keep the critical security components (such as encryption key managers) inside their infrastructure, so they are never deployed in a public cloud. This approach allows having stronger security at a lower price, since the commodity tasks are performed by cloud providers’ teams of experts in public clouds, while the critical data access components, such as encryption keys and user authorization, are deployed in-house. Figure 5 shows an example of SDS being deployed in Hybrid cloud, where only the storage is in the public cloud, and the computation, KMS and SDS are on-prem or in a private cloud.

Figure 5 SDS in Hybrid cloud

The SDS is based on free open source technologies. It supports almost transparent data security, except for some minimal configuration such as KMS connection details, and exposes
standard interfaces and formats. It helps to leverage hybrid cloud and it uses cutting-edge data security mechanisms.

The SDS can run in a containerized environment thus enabling run-time isolation and protection.

Audit logs and security events will be collected by the SDS and sent to Keenaï using the Syslog protocol, as described in the Interfaces section II.6.1.

II.4. Architecture

The following figure presents the High-Level Architecture of the SDS, as it was presented in D2.1:

The Secure Data Service exposes a simple API to the use case, which basically enables to persist and fetch data from storage in a secure way. On "put data" requests, the SDS uses Parquet encryption in order to store data encrypted in an analytics-friendly format partially in the cloud and partially (if at all) on SME premises. On "get data" requests, the SDS uses Parquet encryption in order to read the encrypted data and either grant or deny access to it. The engine behind the SDS is either Apache Spark [19], Spark with Hudi [20], pandas [21] or others, which use Java, C++ or PyArrow parquet packages with Parquet Modular Encryption. The engine is chosen based on the use case. The SDS uses a Key Management Service to manage master encryption keys used in envelope wrapping of data keys that are generated on-the-fly by PME. Data access control is carried out by the Key Management Service based on access tokens provided by use cases when accessing the SDS. Authentication of users is done by KeyCloak [17], as introduced in D2.2 section IV.3.4, so the authentication token generated by KeyCloak is the token used by KMS to grant or deny access to the encryption keys.

Data access events and security alerts are sent to the SIEM for further processing, before being sent to the Service Ledger for secure sharing.

In the first version, by default all columns of the data will be encrypted and the corresponding master encryption keys will be created automatically in KMS. However, in the next version encryption Policies will be explored, for example using the Open Policy Agent (OPA) [9]. We will analyse what would be the best balance between good user experience and high security.
There are two deployment options of SDS, and they are described in the next two sub-sections.

II.4.1. SDS with HTTP API

Figure 7 presents the deployment option of SDS with an HTTP API, where the SDS Gateway is deployed as an HTTP server based on the Play Framework [18], and the SDS engine is Apache Spark or Apache Spark with Hudi or Iceberg table management, based on the use case needs. The SDS engine uses a Spark version that contains the parquet-mr Java implementation of Parquet with PME. The current version of Spark contains an older version of parquet-mr, which does not include PME. Therefore, the SDS uses the version of parquet-mr that is supported by Spark, but the PME that has been backported to the version supported by Spark. In addition, Hudi also uses this parquet-mr version with backported PME.

The use case application sends HTTP queries to the SDS Gateway. The HTTP methods currently exposed by the HTTP API are:

- sql-query – run a Spark SQL query on Parquet files or a Hudi/Iceberg table
- insert-data – insert data into an encrypted Hudi/Iceberg table
- bulk-import – import data from a CSV to an encrypted Parquet file or a Hudi/Iceberg table

In this deployment option, the use case can be in any language and use any HTTP client library in order to communicate with the SDS Gateway using the HTTP (TLS) protocol.

Hashicorp Vault [7] is used to manage master encryption keys, which do not need to leave the premises of the SME, and it manages access control to keys based on access tokens generated by KeyCloak. Transit engine is used in Vault in order to manage master keys, and envelope encryption is used, where data is encrypted using generated data keys, and these data keys are wrapped by the master keys stored in Vault.

The storage can be configured either as an S3 compatible object storage, e.g. MinIO Object Storage [25], or a local file system, or any other storage supported by Apache Spark Datasource API.

On data access events and security alerts, the SDS will use Syslog to send the messages to SIEM, as described in section II.6.1. Interfaces.
II.4.2. SDS deployed as a library

Figure 8 presents the deployment option of SDS deployed as a library, as part of the Use case tooling. The use case application can operate on different python data processing framework, such as Pandas, Modin, TensorFlow, Ray, etc. The SDS engine is wrapped in PyArrow [8] python library which contains PME implementation, and the API is:

- ParquetWriter.write_table for writing encrypted parquet data
- ParquetFile.read() for reading encrypted parquet data

As in the first deployment option, master keys and access to them are managed in Hashicorp Vault Key Management Service, and data access events and security alerts are sent to Syslog.

SDS as a library

![Diagram of SDS deployed as a library](image)

Figure 8 SDS deployed as a library

II.5. Implementation Details

II.5.1. PyArrow with PME

The Java implementation of the Parquet encryption standard has just been released with PME in parquet-mr 1.12 by the Apache Parquet community. Now, the Apache Spark community is working on integrating it in its upcoming release of Spark 3.2. In addition, after the C++ implementation was released in Apache Arrow 4.0.0, IBM Research began working with the Apache Arrow community to expose PME in PyArrow, as well [22]. This will enable Python applications to read and write encrypted Parquet with PyArrow, Pandas, Modin and other frameworks using PyArrow. It involves working with the community on the Python interface to PME and on its ease of use.

II.5.2. Key Management Service

SDS stores data encrypted using Parquet encryption with master encryption keys managed by a Key Management Service (KMS) and data encryption keys generated by Parquet writer, wrapped with the master encryption keys. This way the master encryption keys do not leave
the KMS and the KMS itself can remain on the premises of the SME in a Hybrid Cloud or Private Cloud scenario.

Hashicorp Vault Key Management Service is such a KMS, which enables envelope encryption with its transit secrets engine [23]. Vault does not store the data sent to the secrets engine. It can also be viewed as "cryptography as a service" or "encryption as a service".

In order to use the Vault, the SDS has to receive an access token to pass to Vault when using it to wrap/unwrap the data encryption keys (Figure 9). Based on access policies configured in Vault [24] the wrap/unwrap action will succeed or fail.

As described in the section IV.3.4 of D2.2, the authentication provider considered for the project is KeyCloak. Therefore, JSON Web Token (JWT) authentication has to be enabled in Vault and Vault has to be configured with KeyCloak as its OpenID Connect Identity Provider.

![Figure 9 Hashicorp Vault Key Management Service](image)

### II.5.3. Configuration of PME

In the first version of SDS, the configuration of PME, i.e. which columns in a table to encrypt with which keys, is either automatic, where all columns are encrypted by default, or customised in configuration files of SDS. In the next versions, we intend to look at Open Policy Agent (OPA) for policy configuration and validate whether that simplifies the SME’s experience with the toolkit.

The following PME properties can be configured in SDS:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>parquet.encryption.column.keys</td>
<td>List of columns to encrypt, with master key IDs (see HIVE-21848). Format: <code>&lt;masterKeyId&gt;:&lt;colName&gt;,&lt;colName&gt;;&lt;masterKeyId&gt;:&lt;colName&gt;</code>...</td>
<td>None. If neither column.keys nor footer.key are set, the file won’t be encrypted. If one of the two properties is set, an exception will be thrown.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>parquet.encryption.</td>
<td>Master key ID for footer encryption/signing.</td>
<td>None. If neither column.keys nor footer .key are set, the file won't be encrypted. If one of the two properties is set, an exception will be thrown.</td>
</tr>
<tr>
<td>footer.key</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parquet.encryption.</td>
<td>Parquet encryption algorithm. Can be AES_GCM_V1 or AES_GCM_CTR_V1.</td>
<td>AES_GCM_V1</td>
</tr>
<tr>
<td>algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parquet.encryption.</td>
<td>Write files in plaintext footer mode, that makes many footer fields visible (e.g. schema) but allows legacy readers to access unencrypted columns. The plaintext footer is signed with the footer key. If false, write files in encrypted footer mode, that fully encrypts the footer, and signs it with the footer key.</td>
<td>FALSE</td>
</tr>
<tr>
<td>plaintext.footer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parquet.encryption.</td>
<td>Class implementing the KmsClient interface. &quot;KMS&quot; stands for “key management service”. The Client will interact with a KMS Server to wrap/unwrap encryption keys.</td>
<td>org.apache.parquet.crypto.keytool.VaultClient</td>
</tr>
<tr>
<td>kms.client.class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parquet.encryption.</td>
<td>ID of the KMS instance that will be used for encryption (if multiple KMS instances are available).</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>kms.instance.id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parquet.encryption.</td>
<td>URL of the KMS instance.</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>kms.instance.url</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parquet.encryption.</td>
<td>Lifetime of cached entities (key encryption keys, local wrapping keys, KMS client objects).</td>
<td>600 (10 minutes)</td>
</tr>
<tr>
<td>cache.lifetime.seconds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parquet.encryption.</td>
<td>If true, then footer key and column keys are generated on the fly.</td>
<td>FALSE</td>
</tr>
<tr>
<td>policy.free</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 10 shows an example where different columns of a table are encrypted with different encryption keys, and various tokens are granted access to different master encryption keys. Key1 encrypts Column1 and Column3, Key2 encrypts Column2, and Key3 encrypts Column4. Please, note that because of envelope encryption, the columns are not encrypted directly, but their key encryption keys are encrypted by master keys managed by the KMS. Secure backup script should provide an access token, which grants access to all the keys. User1 should provide an access token, which grants access to keys Key1 and Key2, and User2 should provide an access token, which grants access to key Key1. In addition, all these access tokens should grant access to the footer key, which should be specified by the property parquet.encryption.footer.key. The property parquet.encryption.column.keys would be:

Key1: Column1, Column3; Key2: Column2; Key3: Column4

The KMS URL would be provided in the property parquet.encryption.kms.instance.url.
II. 6. Interfaces

II. 6. 1. SDS Interfaces with other Tools within the CK4SME Toolkit

II. 6. 1. a. SDS <> Keenai

SDS interacts with the SIEM Keenai by sending audit and security events through the Syslog protocol as JSON messages. The goal of this interaction is to enable audit and inspection of data and processing to validate business and regulatory requirements. The following events will be triggered:

- Data integrity violations
- Access to encryption keys for reading encrypted data
- Failed attempts to access encryption keys
- Policy decisions based on regulatory requirements

The JSON message will contain such information as:

- classification, name of source, IP address, identity of caller, table name, query

Since the JSON structure is flexible and extensible, we do not need to commit to all the fields yet, and the field list will be revised.
II.6.2. SDS Interfaces for Use cases

SDS with HTTP API exposes an OpenAPI HTTP endpoint with dynamically generated documentation of the available services, see Figure 11. The OpenAPI Specification [26] defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.

![Figure 11 SDS HTTP API – OpenAPI live documentation](image)

In the following sub-sections, three SDS methods for querying, importing and adding data will be shown. For each method, an OpenAPI specification will be presented, an example cURL call, and an example application that might call the SDS method.

II.6.2.a. SDS HTTP API – SQL Query

The SQL query method enables to run a SQL query on the data defined in one of the tables, either a plaintext table or an encrypted one. The SQL query syntax is the Spark SQL SELECT syntax [27]:

```sql
SELECT [ hints , ... ] [ ALL | DISTINCT ] { [ [ named_expression | regex_column_names ] [ , ... ] | TRANSFORM (...) ] }
FROM { from_item [ , ... ] }
[ PIVOT clause ]
[ LATERAL VIEW clause ] [ ... ]
[ WHERE boolean_expression ]
[ GROUP BY expression [ , ... ] ]
[ HAVING boolean_expression ]
```

The OpenAPI specification:
A cURL example call with SQL query:

```bash
url --location --request POST 'http://localhost:9000/sql-query' \
--header 'Authorization: Bearer 1234567890abcdefg' \
--header 'Content-Type: application/json' \
--data-ran '{ "query" : "SELECT * FROM transactions LIMIT 2" }'
```

Figure 12 OpenAPI specification of SQL query

Figure 13 cURL SQL query
Example application:

![Example application with SQL query](image1)

**Figure 14 Example application with SQL query**

**II.6.2.b. SDS HTTP API – Bulk import**

The Bulk import method enables to do a bulk import from a CSV file into a table, which is either a Parquet file or a Hudi table. If the table is defined as encrypted, then the data will be encrypted on the fly.

![OpenAPI specification of Bulk import](image2)

**Figure 15 OpenAPI specification of Bulk import**
An example cURL call for Bulk import:

```bash
curl --location --request POST 'http://localhost:9000/bulk-import' \
--header 'Authorization: Bearer [TOKEN]' \
--header 'Content-Type: application/json' \
| "targetTable" : "transactions" }'
```

**Figure 16 cURL Bulk import**

Example application:

**Figure 17 Example application with Bulk import**
II.6.2.c. SDS HTTP API – Insert Data

The insert data method can insert data into a Hudi table. If the table is encrypted, the data will be encrypted too.

Figure 18 OpenAPI specification of Insert data

An example cURL call for Insert data:

```bash
1 $url --location --request POST 'http://localhost:9000/insert-data' \
2 --header 'Authorization: Bearer [access_token]' \
3 --header 'Content-Type: application/json' \
4 --data-urlencode "\n5 | "date": { "ts": "3", "uuid": "3", "begin_lat": 0.159, "begin_lon": 0.8108, "driver": "driver-284", "end_lat": 0.983, "end_lon": 8.376, "fare": 29.47, "partitionpath": "iberica/brasil/seo_sao_paulo", "rider": "rider-213" },\n6 | "targetTable": "hudi_trips" }"
```

Figure 19 Example application with Insert data
II.7. Usage

SDS with PME can be used in various use cases in order to manage data in a secure way. Following are some of the possible usages of SDS and PME in use cases proposed by the SME partners of the CyberKit4SME project from two validation fields: Finance and Smart Transportation.

The financial use case scenario (see Figure 21), presents a small financial institute buys Foreign Exchange tick data that records every price change about once per second for every pair of currencies. The financial institute gives orders to traders to buy or sell currencies based on the analytics models that run on the ForEx data. Clearly, confidentiality is important since this detailed data has been paid for, but its integrity is important too since financial decisions are made based on the data. Any missing or erroneous data can affect the decision and possibly result in great financial losses. Moreover, storing the data should be cheap and easy for the SME partner. As a result, SDS can help address the requirements of this use case, since it provides privacy and integrity protection of the data. It is affordable because of the excellent compression of Apache Parquet format, and the performance of analytics queries running on these parquet files is good.

![Figure 21 SDS with HTTP API in a Financial use case](image)

There are additional methods planned to be implemented, such as Delete by identifier.
Another interesting use case for PME is the smart transportation use case, where data is collected from cars (e.g., images, positions, acceleration and velocity) – see Figure 22. This data is used to build and train machine-learning models using TensorFlow, which are then used in smart cars to make real-time decisions.

The data collected from the cars contains sensitive information, so it must be stored in a way that is compact and encrypted. That said, various personas should be able to run Python scripts on this data to analyse it and to train models. PME allows storage of large amounts of data in a compact way encrypted with different encryption keys (e.g., according to sensitivity levels) and to give access to the encryption keys based on security clearance or some other enterprise policy. Access control is achieved by controlling access to the keys without creating multiple replicas of the table — the physical data files remain accessible to a large set of people, but they can only read data for which they have access to keys. For example, Figure 22 presents a scenario, in which two different users run queries on the same table that has five columns encrypted with PME. The first user selects three columns out of the four available to them based on permissions granted with their access token, and the second user selects two columns out of the three available to them based on their access token. That might be achieved by using one key for the least sensitive columns 1, 3 and 5, another key for the more sensitive column 2 and yet another key for the most sensitive column 4.
II.8. Validation

This section describes the features delivered to WP3 for the definition of validation use case scenarios. The validation scenarios themselves are described in D3.2, where each SME partner has a section with the description of each scenario.

The SDS can be used:

- As a database engine: data insert / delete / get / query
- For getting data from standard sources (streams, and file formats like CSV etc)
- For encrypted export/import of bulk data
- For anonymized export of bulk data
- For cloud backup and ransomware protection
- For deleting individual records (e.g., for GDPR)
- Both for data and/or metadata

Ransomware protection will be investigated in later stages of the project, and anonymization might be a stretch goal.

Properties of SDS:

- It is based on free open source technology, so no download or maintenance charges are incurred
- It is easy to use
- It supports (almost) transparent data security
- It exposes standard interfaces and formats
- It helps to leverage hybrid cloud
  - SME IT tasks can be offloaded to public cloud to cut expenses and to lower maintenance costs
  - No vendor lock-in – if standard technologies are used, the SME can switch to any public cloud
  - Extra-sensitive data can be kept on-premises, while larger-scale sensitive data can be moved to the cloud
- It uses cutting-edge data security mechanisms
  - Contribution to leading open source repositories as part of the CyberKit4SME project
  - Work on integrating the technologies in public clouds (leveraged by various customers, including SMEs)

The validation scenarios will validate, in addition to the functionality, also the ease of deployment and the documentation.
II.9. Next Steps

This section provides a detailed roadmap of the next steps of SDS implementation. This roadmap follows the technical coordination Gantt chart of the project and details the implementation steps that will follow in the next months. The timeline of this work starts at M17 of the project and will proceed until M34.

The roadmap is divided into five main tasks as shown in Figure 23, describing the SDS functionalities that will be implemented for the CyberKit4SME project. The tasks and their schedule are organised as follows:

**M17-M24**: Implementation of Parquet Modular Encryption in PyArrow – first at file-level, and then at dataset level. This task will continue in parallel to some other tasks, since it involves Open source collaboration and is awaiting community feedback.

**M17-M18**: Integration of SDS with Keenai, generation of audit and security events.

**M19-M20**: Integration of SDS with User Management System, which is KeyCloak in this project.

**M21-M28**: Dataset integrity protection – work with the OSS community on table-format integrity protection, either Hudi or Iceberg.

**M28-M34**: Further Implementations eventually identified with the validation scenarios, including policy management and governance, ransomware protection and any other.

We plan to release the SDS as a Python library for validation at M24 of the project.

An important goal that will be considered in all the tasks is ease of deployment and usage in order to provide good usability to the SMEs that don’t have the resources that large enterprises have to invest in experts. The documentation and ease of deployment will be assessed by the Use case partners in WP3 during validation sprints. We will also try and work on deployment automation as much as possible where it makes sense.

<table>
<thead>
<tr>
<th>SDS Roadmap</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>PME in PyArrow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- File-level PME</td>
<td>M17</td>
<td>M24</td>
</tr>
<tr>
<td>- Dataset-level PME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration with Keenai</td>
<td>M17</td>
<td>M18</td>
</tr>
<tr>
<td>Integration with User Management</td>
<td>M19</td>
<td>M20</td>
</tr>
<tr>
<td>Dataset integrity protection</td>
<td>M21</td>
<td>M28</td>
</tr>
<tr>
<td>Further Implementation:</td>
<td>M28</td>
<td>M34</td>
</tr>
<tr>
<td>- Policy management and Governance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Ransomware protection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Figure 23 SDS Roadmap |
III. SECURITY INFORMATION SHARING AND COLLABORATION FRAMEWORK

III.1. Background

CyberKit4SME will provide a framework for secure, privacy-aware information sharing based on blockchain technology. This framework is implemented into the Service Ledger (SL) tool developed by the UoS Cyber. SL is a Blockchain-as-a-Service platform offering programmable blockchain-empowered services encompassing privacy-preserving data sharing, distributed access control, and non-repudiable data computation.

In the CyberKit4SME project, SL will allow SMEs and MEs to share security intelligence in a fully decentralised network by integrating standards for data representation and transport protocols into a blockchain-based data-sharing platform supported by a secure and privacy-preserving decentralised data storage system.

III.1.1. CTI Sharing: STIX and TAXII standards

Cyber Threat Intelligence (CTI) provides information about threats and malicious actors related to cyber-attacks. The organisations can use CTI for correlation within their security systems with their security systems to mitigate harmful events in cyberspace enhancing their monitoring, detecting and preventing malicious activities and consequently improving their security resilience. CTI includes technical intelligence such as Indicators of Compromise (IoCs) (e.g. IP addresses, virus signatures, file hash of malware, URLs or domain names) or exploits, and operational and strategic intelligence like Tactics Techniques and Procedures (TTPs) of threat actors. TTPs give a precise overview of the adversaries’ behaviours and are used to spot the actor’s tendency to use certain malware variants, attack tool, delivery mechanism (e.g. phishing), or exploit.

Government, institutions and private organisations can share CTI indicators to improve awareness and responsiveness to cyber-attacks by correlating and analysing information from multiple sources. CTI sharing allows an organisation to enhance the detection and prediction of cyber threats or ongoing cyber-attacks that target a particular sector or community. To facilitate the exchange of CTI, the information needs to be represented and shared adopting standardised data format and transport protocols. The use of standards allows different organisations, repositories of CTI and tools to share data autonomously and at machine speed, without the need of human intervention.

The Structured Threat Information eXpression (STIX) [28] is a data modelling language for the representation and serialisation of CTI in a structured way. STIX is nowadays employed in several data-sharing tools and CTI repositories, representing a de-facto standard for CTI. STIX is based on JSON and defines a taxonomy for representing CTI as a collection of JSON objects, viz. STIX bundle. The latest version of the standard, viz. STIX2.1, defines three categories of objects as STIX Domain Objects (SDO), STIX Cyber-observable Objects (SCO) and STIX Relationship Objects (SRO).

- **STIX Domain Objects**: SDOs are operational and strategic intelligence objects representing behaviours and constructs defining an attack landscape (e.g. attack patterns, identities, malware, vulnerabilities)
- **STIX Cyber-observable Objects**: SCOs are technical intelligence representing observed facts about a network or a host that can be related with SDOs to better represent an attack landscape (e.g. IP addresses, files, processes)
- **STIX Relationship Objects**: SROs connect SDOs and SCOs together to better represent the attack landscape defining the type of association between objects
In CTI sharing, STIX-represented CTI are exchanged over HTTPS via a standardised data exchange protocol known as *Trusted Automated Exchange of Intelligence Information (TAXII)* [29]. Specifically, TAXII defines a set of standard client-server APIs establishing common CTI sharing models based on two concepts, viz. *Collections* and *Channels*. Figure 24 illustrates the communications between the TAXII server and clients interacting via Collections and Channels:

- **TAXII Collections**: Represent the interface to a logical repository of CTI hosted by a TAXII server and accessed by a TAXII client to send or request information. Collections are used to exchange information in a request-response manner;
- **TAXII Channels**: Hosted by a TAXII server, permit TAXII clients to exchange messages with other TAXII clients in a publish-subscribe model. TAXII clients publish messages to channels and subscribe to channels to receive published messages.

Collections and Channels are organised by a TAXII server under a certain *API Root* supporting trust subgroups and other categories.

### III.1.2. The Blockchain Technology

The *blockchain* is a prominent technology emerged in the recent years as a mean of governance in fully decentralised systems composed by many entities that do not trust each other [30]. Specifically, the blockchain distributes trust over a peer-to-peer (p2p) network and governs the operations between parties avoiding any centralised, trusted authority. For example, the participants of a decentralised system might want to directly collaborate by exchanging information, accessing services and consuming data, under untrustworthy conditions and without relying on third parties. Blockchain offers trustworthiness in p2p networks by executing a consensus protocol between participants that commonly agree on the operations (and their order) between two or more parties.

Practically, a blockchain is a decentralised ledger fully replicated over the participants of the network that collects transactions happening between parties into cryptographically secured linked blocks - Figure 25. Each block points at its predecessor, collects the state of the blockchain (version, timestamp, general information), and a list of transactions. The data structure is a read-only/write-once in which information is securely stored and impossible to be reverted. Therefore, the information stored into the blockchain is immutable and tamper-proof; the history of transactions cannot be reversed unless the attacker controls the majority of the network. In addition, blockchains guarantee the traceability of operation, making decentralised systems verifiable and trustworthy.

Blockchains firstly appeared in Bitcoin as distributed ledgers storing financial exchanges of cryptographic assets, e.g. Bitcoin cryptocurrency [31]. These systems rely on public, permissionless networks, viz. permissionless blockchains, where everyone is allowed to
participate. Permissionless blockchains are extremely secure thanks to the high level of decentralisation; once a transaction is stored into the blockchain it is replicated over a large number of nodes, hence an attacker that aims at corrupting the blockchain must subvert a majority of the network (51% attack) which is highly improbable. Differently, in application scenarios where decentralisation (hence security) is not the first requirement, blockchains networks can be integrated with an authentication layer to establish trust, viz. permissioned blockchains. Permissioned blockchains are networks of a fixed size where participants know each other. These networks are more efficient and enhance privacy at the expense of security; the lower degree of decentralisation makes the network easier to corrupt.

Figure 25 – Blockchain data structure.

In recent years, blockchains evolved to more complex and powerful systems thanks to the integration of smart contracts, viz. fully decentralised programs stored and executed directly on the blockchain. Typically, smart contracts are used to regulate operations involving several participants and to validate their execution according to predefined conditions. Smart contracts make blockchains programmable and enable the realisation of blockchain-based systems for novel use case scenarios such as decentralised applications (dApps), decentralised financial services, data sharing and tokenisation of digital assets as fungible and non-fungible tokens.

Nowadays there exist several blockchain platforms offering the possibility of realising a large number of different applications. In the context of CyberKit4SME, we will use the prominent blockchain platform called Algorand [32]. Algorand blockchain proposes itself as the most reliable, efficient and secure blockchain platform ensuring transaction throughput comparable to the well-established financial networks. Algorand takes advantage of a new consensus protocol, i.e. Pure Proof of Stake (PPoS) can guarantee agreement among a fully decentralised network, without giving up on scalability or security. In addition, Algorand provides a complete toolkit of functionalities including but not limited to smart contracts and representation of digital assets, that enables the realisation of flexible and efficient applications.

### III.1.3. Decentralised Data Storage

In classical client-server architectural patterns, the systems store data into centrally located servers managed by single organisations or cloud providers. Differently, decentralised data storage (DDS) systems store files across a peer-to-peer network of multiple servers operated by single users. In DDS, each file is divided into fragments, viz. Shards, and stored encrypted across the DDS network. The data owners are the only ones entitled to reconstruct and decrypt their data by querying the nearby peers. The principal advantages of using DDS are:

- Security and Privacy: There is not any centralised entity holding the information. Data owners remain in control of their data, limiting any issue of data outages and data breaches. Therefore, DDS guarantees privacy over stored information since shard holders are not able to reconstruct the information nor to decrypt it;
• High transfer speed: Data is always stored and retrieved through the nearby peers of a distributed network;
• High availability: Data is replicated across multiple peers and does not depend on single server availability. Replication guarantees users have access to data, even in case of server faults or attacks.

Recently, several applications combined DDS systems with blockchains to improve the poor storage capabilities of the blockchains. Indeed, blockchains cannot store a large amount of data because it would cause too large blocks, hard to be propagated, that may lead to performance and scalability issues. However, combining blockchains with DDS permits to preserve the decentralisation and security offered by blockchains while guaranteeing efficient data storage.

Recently, several DDS platforms emerged proposing thus different types of decentralised networks. In the context of CyberKit4SME, we consider the following platforms:

• **IPFS** [33]: InterPlanetary File System (IPFS) is a protocol for storing and accessing files from a fully decentralised network. When a new file is added to IPFS it is split into cryptographically secured shards, and assigned with a unique fingerprint called Content Identifier (CID). CIDs uniquely identify data and are used in IPFS to retrieve information from the network.

• **Solid Project** [34]: Solid is a DDS protocol allowing users to store data into a decentralised network of nodes, called *pods*. Pods are owned by data producers and can be configured to grant or revoke access to stored data according to permission rules. Data can be accessed by authenticated users and applications connecting to a certain pod using a unique ID. Data remain stored into the pods, safely controlled by data owners.

### III.2. Goals

The goal of the Security Information Sharing and Collaboration Framework is to equip SMEs and MEs with an online collaborative system through which to securely share cybersecurity information with CERTs and sectorial supply chain partners. Such a system will enhance the risk monitoring activities with a larger knowledge base of data and facilitate the interaction between SMEs/MEs and CERTs while guaranteeing privacy and business confidentiality. This framework will provide a means for SMEs/MEs to actively share security information engaging in a collaborative response to cyber security, improving the situational awareness of cyber threats, facilitate preparedness and responses to cyber-attacks and take a more active role in a stronger, collective approach to security, privacy and data protection within their sector. In addition, it will support SMEs and MEs to meet the NIS directive regulatory obligation of reporting security incidents to CERTs, preserving the privacy of the source and avoiding the exposition to possible reputation damage.

The goals of the Security Information Sharing and Collaboration Framework can be summarized as follows:

• Enable the sharing of CTI information related to threats detected by the monitoring system and security events/alerts with CERTs and sectorial supply chain partners, while preserving the security of shared information and privacy;
• Facilitate SMEs and MEs to comply with the mandatory incidents reporting with CERTs, enhancing the preparedness and responses to cyber-attacks for critical sectors while preserving the source business confidentiality.

To accomplish these goals, we combine the blockchain technology and DDS with standard attribute-based access control rules. In particular, the blockchain technology allows SMEs and
MEs to collaborate over a decentralized network and share CTI in a secure manner. Simultaneously, access control rules enable the sharing with sectorial partners enhancing privacy and business confidentiality. In addition to that, the sharing framework will build on common standards for CTI representation and exchange, to ensure full interoperability with other systems and tools. The traceability of blockchain sharing protocols and the integration with cyber threat intelligence standards will allow CERT/CSIRT to trust threat intelligence and disseminate recommended actions across the supply chain to timely inform SMEs/MEs on the most appropriate mitigation and remediation strategies for emerging threats.

III.3. Overview

The Service Ledger (SL) is a blockchain-as-a-service platform offering programmable blockchain-enabled services that can be adapted to several application scenarios. In CyberKit4SME, SL enables the sharing of CTI between different organisations and local authorities such as CERTs and CSIRTs while preserving the security of shared information and privacy of data sources. The organisations improve the awareness and the defence against threats and malicious activities by using the information shared by others and actively participate with the platform by sharing their CTI coming from their monitoring and risk assessment systems.

SL ensures security and data sharing using a novel approach that separates computation from persistency. Specifically, in SL, the CTI is physically stored on a DDS and simultaneously represented as a unique, immutable, digital asset on the blockchain. The digital asset is assigned to a blockchain identity, i.e. the data owner, and eventually shared with other participants according to defined rules and conditions governed by a smart contract. Digital tokens stored on the blockchain are immutable, tamper-proof, and verifiable representations of data shared on DDS. A blockchain-based data-sharing platform adopting such an approach achieves performance improvements (the blockchain is used as immutable registry of transactions and not as a database), better data availability (thanks to the DDS), improved security and privacy guarantees. Therefore, SL enables interoperability with other systems and sharing tools, embedding TAXII and STIX standards for the representation and exchange of CTI. The use of standards facilitates the interaction with current legacy systems without requiring additional implementations or data parsing activities.

In addition, SL allows organisations to join sectorial sharing communities, e.g. supply chain partners and local authorities, allowing the security intelligence sharing while preserving privacy and business confidentiality. Specifically, SL embeds attribute-based encryption sharing protocols [35] allowing users to define ad-hoc, private, sharing groups, manage the visibility of their data and the access control rules, or even share CTI anonymously. The encryption keys will be managed by a dedicated Key Management System [7]. The distribution of sensitive intelligence will be based on data classification based on Traffic Light Protocol [36].

Figure 26. Overview of the Service Ledger Instance in CyberKit4SME
Figure 26 illustrates the overview of an SL instance into CyberKit4SME. The figure shows the components interacting with SL and all the modules composing an SL instance. Particularly, SL enables data producers/consumers such as the SMEs/MEs and the CERTs to access the blockchain (either permissionless or permissioned) and DDS for their CTI sharing activities exposing configurable services defined with the following modules:

- **Configuration Manager**: Software module implementing the connector to DDS, blockchain and Key Management System components. It defines the connection and configuration parameters;
- **Blockchain API and Orchestrator**: These two modules are in charge of managing the interactions between SL and the blockchain native services like creating an asset on a permissionless or permissioned blockchain, sending a transaction or creating/deleting a blockchain account;
- **DDS API**: Software module integrating DDS native services into SL, like storing/retrieving a file;
- **TAXII**: A TAXII server defining the standard APIs exposes by SL to share CTI;
- **Access Control and Privacy**: Characterised by three components (i) Membership Manager, (ii) Data Access Control Manager, (iii) Privacy Manager. The former two components are in charge of managing SL users and the access control rules on SL resources and services, the latter implements privacy functionalities like anonymous transactions.

The TAXII server component of SL represents the direct interface to CTI producers/consumers, also called TAXII clients. In the context of the CyberKit4SME toolkit, three TAXII clients will access SL services to share CTI like IoC or other cybersecurity-related information such as security alerts and incident reports. In principle, the SL will interact with the other tools below mentioned, but in the future, we do not exclude other interaction directly from the SMEs:

- **Risk monitoring and management tool - Keenaï SIEM**: The SIEM tool of SMEs/MEs will be interested in retrieving sectorial CTI to increment its knowledge base and share monitored threats. In addition, Keenaï will be interested to share security alerts or incident reports;
- **Risk assessment tool – SSM**: Data from SL will be conveyed to Keenaï and potentially from there to SSM. A direct link between SSM and SL (whether through technical or human integration) may be defined later, as the usage of the tools by the SME partners in the validation proceeds;
- **CERTs/CSIRTs authorities**: collect and distributed detected CTI with the whole network or with specific sectors.
III.4. Architecture

The Service Ledger (SL) is a blockchain-enabled platform offering programmable services that directly (or indirectly) interact with one or more decentralised systems. In CyberKit4SME, SL enables SMEs and MEs to collaborate and share CTI over a fully decentralised network avoiding data being stored on centralised providers.

![Diagram of SL CTI sharing platform architecture]

Figure 27 shows an overview of the SL sharing platform and its network. The Organisations (SMEs and MEs) and the Local Authorities (CERT and CSIRT) engage in a collaborative platform sharing CTI and security events (e.g. security alerts or incident reports) represented as STIX objects, and exchanged through TAXII standard transport protocols implemented in SL. SL acts like a TAXII Server offering secure data sharing and privacy preserving functionalities. SL relies on a decentralised repository (DR) that, differently from classic CTI sharing platforms based on centralised repositories, stores data in a fully decentralised system composed of a DDS linked with a blockchain. Data is stored on a DDS and simultaneously linked to a unique digital token minted on the blockchain. All the CTI stored into the DR benefits of strong immutability properties, it is tamper-proof and always verifiable. The blockchain identified to implement the DR is Algorand, because of its decentralisation, security and scalability guarantees [37]. Accordingly, the choice of a DDS is strictly related to the interoperability with the Algorand blockchain. We identified two DDS platforms, such as IPFS and Solid, and a choice between them will be taken after an evaluation of their reliability and security.

Organisations and Local Authorities join the sharing platform via an SL instance (either running on-premises or in other trusted environments). Each SL instance embeds authentication and authorisation features in order to identify users of different organisations/authorities and expose them to their restricted resources. The management of users and their roles is still

---

1 Minting is the process of token creation on a blockchain system
under evaluation; however, we are considering the integration of a federated authentication mechanism to be used for the whole CyberKit4SME toolkit.

Figure 28 illustrates the technical architecture of a single SL instance and the building blocks composing the software. Each SL instance runs within a container-based environment – Docker [38], and includes the following components:

- **TAXII Server**: A web server complying with the latest TAXII Collection Server Requirements [39]. This component exposes the APIs to be invoked by TAXII Clients for sharing and retrieving CTI from SL;

- **SL Server Application**: Core SL’s backend application implementing the connectors with the DDS and blockchain systems. It serves the TAXII Server requests retrieving/sending data to the TAXII DR. In addition, this component implements the access control and privacy functionalities integrating a Data Encryption Module. Finally, it integrates a database for storing information related to users and the custom parameters used by the blockchain and DDS connectors;

- **SL Vault**: Secure encryption keys storage vault used to store encryption keys of particular sharing groups;

- **SL Client Application**: Frontend user application that can be used to access SL’s services. It will expose REST APIs to interact with the data-sharing platform. In addition, we are considering implementing a web-based client interface in which users can access via the browser;

- **Algorand Node**: Blockchain node running within SL and connected to the Algorand Blockchain Network. This node stores the Blockchain wallet and the encryption keys associated with users. A blockchain wallet is a collection of blockchain identities; the encryption keys are used to generate those identities and interact with the blockchain; blockchain identities are associated with SL users. The Algorand node represents the entry point to the Algorand Blockchain Network and is used to access the (i) Transactions Ledger; (ii) the community public keys representing the blockchain addresses of the participants, (iii) the smart contracts. The SL server application will integrate the Algorand SDK to directly interact with the blockchain functionalities;

- **DDS Node**: Node running within SL and connected to a DDS network. All the shared data will be stored on DDS via such a node using its embedded APIs.

SL will deploy a decentralised application (dApp) on the Algorand Blockchain Network as a smart contract. Such dApp will control the operation of CTI digital token creation and exchange.

---

2 A blockchain address is the public key generated with asymmetric encryption schema
III.5. Implementation Details

SL will be deployed as a multiservice containerised architecture implemented with Docker. Specifically, there will be a Docker Compose file configuring the SL architecture. This docker-compose file will include seven independent containers running the following software components:

- **SL Client Application** container running the frontend application accessible via web browser implemented in Angular (or React);
- **SL Server Application** container running a NodeJS server (we leave open the possibility of migrating to Django for compatibility with other tools and CTI standards) implementing the SL backend server. This server will expose the REST API endpoints to SL services;
- **TAXII Server** exposing standard TAXII APIs compliant with the TAXII 2.1 specs. This server communicates CTI straightforward to the application server. CTI are represented as STIX objects compliant with STIX 2.1 specs;
- A PostgreSQL database for storing information used by the SL Server Application;
- **SL Vault** container running Hashicorp Vault [7] to manage encryption keys. We are considering replacing this component with the IBM SDS vault service;
- **Algorand Node** using the Algorand sandbox environment [40]. This component includes the entire Algorand development toolkit and is based on docker;
- **DDS node docker image** (under investigation between IPFS and Solid).

III.6. Interfaces

The SL interface is designed as a TAXII server exposing the REST API endpoints that enable authorised users to query TAXII Collections of CTI objects stored into the decentralised repository (DR). The SL interface only supports communications between TAXII Clients and the SL TAXII Server endpoints encrypted using TSL.

**SL Collections** are organised into **SL API Roots** endpoints. An API Root is an instance of SL APIs available at a certain URL and represents the "root" URL of that particular API. Organising SL Collections into SL API Roots allows SL to guarantee the separation of content and the implementation of access control rules for specific sharing groups. In SL there are as many API Roots as sharing groups, and to any SL API Root may correspond several collections. A sharing group could be a sectorial supply chain of SMEs or a private group between certain SMEs and CERTs. The validation scenarios will define the sharing groups.

The Discovery endpoint displays the list of active API Roots in the SL and exposes information on the SL TAXII Server to authorised users. Figure 29 summarises all the endpoints characterising the SL interface.

Examples of SL API Root URLs are:

- https://sltaxii.cyberkit4sme.eu/sharegroup1/
- https://sltaxii.cyberkit4sme.eu/sharegroup2/

---

3 The TAXII Server and the SL Server Application might be combined in future deployments
In the following subsections we detail the SL endpoints (URLs and HTTP Methods), the supported resources type, and the implementation status – design/in-progress/completed:

- **design**: The API is under definition and can be changed according to validation scenarios;
- **in-progress**: The API has been defined and under development;
- **completed**: The API has been released.

### III.6.1. SL Discovery Service

**Implementation Status**: *in-progress*

This section details the SL Discovery endpoints summarised in Table 2.

#### Table 2 – SL Discovery endpoints

<table>
<thead>
<tr>
<th>URL</th>
<th>Methods</th>
<th>Resource Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/sltaxii2/</code></td>
<td>GET</td>
<td>discovery</td>
</tr>
<tr>
<td><code>{api-root}/</code></td>
<td>GET</td>
<td>api-root</td>
</tr>
</tbody>
</table>

#### III.6.1.a. SL TAXII Server Discovery

This endpoint is defined in Table 3. It provides general information about the SL TAXII Server including the SL API Roots available.

#### Table 3 - SL Discovery API

<table>
<thead>
<tr>
<th>REQUEST</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GET <code>/sltaxii2/</code></td>
<td></td>
</tr>
<tr>
<td>HEADER Accept: application/taxii+json;version=2.1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESPONSE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>200 – Successful Request</td>
<td></td>
</tr>
<tr>
<td>Content-Type: application/taxii+json;version=2.1</td>
<td></td>
</tr>
<tr>
<td>Payload: discovery</td>
<td></td>
</tr>
</tbody>
</table>
FAILURE RESPONSES

401 - The client needs to authenticate
403 - The client does not have access to this resource
404 - The Discovery service is not found, or the client does not have access to the resource
406 - The media type provided in the Accept header is invalid

Content-Type: application/taxii+json;version=2.1
Payload: error

The discovery resource type is a JSON containing the information about the SL TAXII Server and is defined in Table 4.

Table 4 – Discovery resource type.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title (required)</td>
<td>string</td>
<td>A human readable plain text name used to identify the SL TAXII Server.</td>
</tr>
<tr>
<td>description (optional)</td>
<td>string</td>
<td>A human readable plain text description of the SL TAXII Server.</td>
</tr>
<tr>
<td>contact (optional)</td>
<td>string</td>
<td>The human readable plain text contact information and/or the administrator of the SL TAXII Server.</td>
</tr>
<tr>
<td>default (optional)</td>
<td>string</td>
<td>The default SL API Root. Absence of this property indicates that there is no default SL API Root.</td>
</tr>
<tr>
<td>api_roots (optional)</td>
<td>list of string</td>
<td>A list of URLs that identify known SL API Roots. This list MAY be filtered on a per-client basis.</td>
</tr>
</tbody>
</table>

The error resource type is returned by the SL TAXII Server when an HTTP error status occurs. It contains more information describing the error, and is defined in Table 5.

Table 5 – Error resource type.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title (required)</td>
<td>string</td>
<td>A human readable plain text title for this error.</td>
</tr>
<tr>
<td>description (optional)</td>
<td>string</td>
<td>A human readable plain text description that gives more details about the error.</td>
</tr>
<tr>
<td>error_id (optional)</td>
<td>string</td>
<td>An identifier for this error.</td>
</tr>
<tr>
<td>error_code (optional)</td>
<td>string</td>
<td>The error code for this error type.</td>
</tr>
<tr>
<td>http_status (optional)</td>
<td>string</td>
<td>The HTTP status code relative to this error.</td>
</tr>
<tr>
<td>external_details (optional)</td>
<td>string</td>
<td>A URL that points to additional details.</td>
</tr>
<tr>
<td>details (optional)</td>
<td>dictionary</td>
<td>A key:value dictionary – JSON struct - of strings that specifies additional SL TAXII Server specific details about this error.</td>
</tr>
</tbody>
</table>
III.6.1.b. GET SL API Root

This endpoint is defined in Table 6. It provides general information about a SL API Root. The API Roots are often dedicated to a single sharing group.

Table 6 – SL API Root API

<table>
<thead>
<tr>
<th>REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /{api-root}/</td>
</tr>
<tr>
<td>URL PARAM {api-root} – Base URL of the SL API Root</td>
</tr>
<tr>
<td>HEADER Accept: application/taxii+json;version=2.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESPONSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 – Successful Request</td>
</tr>
<tr>
<td>Content-Type: application/taxii+json;version=2.1</td>
</tr>
<tr>
<td>Payload: {api-root}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FAILURE RESPONSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>401 - The client needs to authenticate</td>
</tr>
<tr>
<td>403 - The client does not have access to this resource</td>
</tr>
<tr>
<td>404 - The SL API Root is not found, or the client does not have access to the resource</td>
</tr>
<tr>
<td>406 - The media type provided in the Accept header is invalid</td>
</tr>
<tr>
<td>Content-Type: application/taxii+json;version=2.1</td>
</tr>
<tr>
<td>Payload: error</td>
</tr>
</tbody>
</table>

The api-root resource type is a JSON containing the information about the SL TAXII Server and is defined in Table 7.

Table 7 – API Root resource type.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title (required)</td>
<td>string</td>
<td>A human readable plain text name used to identify this SL API Root.</td>
</tr>
<tr>
<td>description (optional)</td>
<td>string</td>
<td>A human readable plain text description of this SL API Root.</td>
</tr>
<tr>
<td>versions (required)</td>
<td>list of string</td>
<td>The list of TAXII versions that this API Root is compatible with.</td>
</tr>
<tr>
<td>max_content_length (required)</td>
<td>list of string</td>
<td>The maximum size of the request body in octets (8-bit bytes) that the server can support.</td>
</tr>
</tbody>
</table>
III.6.2. SL Collections Service

Implementation Status: design

SL Collections enable the exchange of CTI data between TAXII Clients of a particular sharing group. SL Collections are hosted in the context of a SL API Root. This section defines the SL Collection endpoints summarised in Table 8.

Table 8 – SL Collection endpoints

<table>
<thead>
<tr>
<th>URL</th>
<th>Methods</th>
<th>Resource Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>{api-root}/collections/</td>
<td>GET</td>
<td>collections</td>
</tr>
<tr>
<td>{api-root}/collections/{id}/</td>
<td>GET</td>
<td>collection</td>
</tr>
<tr>
<td>{api-root}/collections/{id}/objects/</td>
<td>GET/POST</td>
<td>envelope</td>
</tr>
<tr>
<td>{api-root}/collections/{id}/objects/{object-id}/</td>
<td>GET</td>
<td>envelope</td>
</tr>
<tr>
<td>{api-root}/collections/{id}/objects/{object-id}/versions/</td>
<td>GET</td>
<td>versions</td>
</tr>
</tbody>
</table>

III.6.2.a. GET SL Collections

This endpoint is defined in Table 9. It provides general information about all SL Collections hosted by an SL API Root and their id which is used to request single Collections and their objects.

If a client fails to authenticate or is not authorised to access this endpoint, it receives an HTTP 401 (Unauthorised) error.

Table 9 – SL Collections API

<table>
<thead>
<tr>
<th>REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /{api-root}/collections/</td>
</tr>
<tr>
<td>URL PARAM {api-root} – Base URL of the SL API Root</td>
</tr>
<tr>
<td>HEADER Accept: application/taxii+json;version=2.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESPONSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 – Successful Request</td>
</tr>
<tr>
<td>Content-Type: application/taxii+json;version=2.1</td>
</tr>
<tr>
<td>Payload: collections</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FAILURE RESPONSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>400 - The SL server did not understand the request</td>
</tr>
<tr>
<td>401 - The client needs to authenticate</td>
</tr>
<tr>
<td>403 - The client does not have access to this collection resource</td>
</tr>
<tr>
<td>404 - The SL API Root is not found, or the client does not have access to the resource</td>
</tr>
<tr>
<td>406 - The media type provided in the Accept header is invalid</td>
</tr>
<tr>
<td>Content-Type: application/taxii+json;version=2.1</td>
</tr>
<tr>
<td>Payload: error</td>
</tr>
</tbody>
</table>

The collections source type is a list of collection resource defined in section III.6.2.b. If no collections are present it returns an empty object.
III.6.2.b. GET an SL Collection

This endpoint is defined in . It provides general information about a single SL Collection hosted by an SL API Root.

If a client fails to authenticate or is not authorised to access this endpoint, it receives an HTTP 401 (Unauthorised) error.

**REQUEST**

<table>
<thead>
<tr>
<th>GET /{api-root}/collections/{id}/</th>
</tr>
</thead>
</table>

**URL PARAM**

- `{api-root}` – Base URL of the SL API Root
- `{id}` – unique collection identifier

**HEADER**

- `Accept: application/taxii+json;version=2.1`

**RESPONSE**

- `200 – Successful Request`
- `Content-Type: application/taxii+json;version=2.1`
- `Payload: collection`

**FAILURE RESPONSES**

- `400 - The SL server did not understand the request`
- `401 - The client needs to authenticate`
- `403 - The client does not have access to this collection resource`
- `404 - The SL API Root is not found, or the client does not have access to the resource`
- `406 - The media type provided in the Accept header is invalid`

- `Content-Type: application/taxii+json;version=2.1`
- `Payload: error`

The collection resource type is a JSON containing the information about an SL Collection and the client read/write permissions on it. The object is defined in Table 10.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id (required)</td>
<td>integer</td>
<td>The id property that universally and uniquely identifies this Collection.</td>
</tr>
<tr>
<td>title (required)</td>
<td>string</td>
<td>A human readable plain text name used to identify this Collection.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>A human readable plain text description of this Collection.</td>
</tr>
<tr>
<td>alias (optional)</td>
<td>string</td>
<td>A human readable collection name that can be used on systems to alias a collection ID.</td>
</tr>
<tr>
<td>can_read (required)</td>
<td>boolean</td>
<td>Indicates if the requester can read (i.e., GET) objects from this Collection. If true, users are allowed to access the Get Objects or Get an Object endpoints for this Collection. If false, users are not allowed to access these endpoints.</td>
</tr>
</tbody>
</table>
### III.6.2.c. GET SL Objects

This endpoint is defined in Table 11. It retrieves objects from an SL Collection. Clients can retrieve single objects of an SL Collection, retrieve all the objects, or paginate through the objects – the details of pagination will be discussed throughout the validation scenarios. Clients may be interested in retrieving objects with a specific id, of a specific type, or with a specific version.

If a client fails to authenticate or is not authorised to access this endpoint, it receives an HTTP 401 (Unauthorised) error.

If the Collection specifies can_read as false for a particular client, this endpoint returns an HTTP 403 (Forbidden) error.

The SL Objects represent STIX 2 elements stored into the DR and are delivered by the SL TAXII Server as envelope resource type. In the case of empty SL Collections, the endpoint returns an empty envelope.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>can_write (required)</td>
<td>boolean</td>
<td>Indicates if the requester can write (i.e., POST) objects to this Collection. If true, users are allowed to access the Add Objects endpoint for this Collection. If false, users are not allowed to access this endpoint.</td>
</tr>
<tr>
<td>media_types (optional)</td>
<td>list of string</td>
<td>A list of supported media types for Objects in this Collection.</td>
</tr>
</tbody>
</table>

**Table 11 – SL GET Objects API**

**REQUEST**

GET /{api-root}/collections/{id}/objects/

URL PARAM

- `{api-root}` - Base URL of the SL API Root
- `{id}` - unique collection identifier

URL FILTERING PARAMS - To be designed

HEADER

Accept: application/taxii+json;version=2.1

**RESPONSE**

200 – Successful Request

Content-Type: application/taxii+json;version=2.1

Payload: envelope

**FAILURE RESPONSES**

400 - The SL server did not understand the request

401 - The client needs to authenticate

403 - The client does not have access to this collection resource

404 - The SL API Root is not found, or the client does not have access to the resource

406 - The media type provided in the Accept header is invalid

Content-Type: application/taxii+json;version=2.1

Payload: {api-root}error
The envelope resource type is a wrapper for STIX2 content. It is used by the SL TAXII Server to exchange STIX2 objects over HTTP. The resource is defined in Table 12.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>more (optional)</td>
<td>boolean</td>
<td>This property identifies if there is more content available based on the search criteria. The absence of this property means the value is false.</td>
</tr>
<tr>
<td>next (optional)</td>
<td>string</td>
<td>This property identifies the server provided value of the next record or set of records in the paginated data set. This property is related to the pagination function under design.</td>
</tr>
<tr>
<td>objects (optional)</td>
<td>list of &lt;STIX Object&gt;</td>
<td>This property contains one or more STIX Objects. Objects in this list MUST be a STIX Object.</td>
</tr>
</tbody>
</table>

### III.6.2.d. Add SL Objects

This endpoint is defined in Table 13. It adds objects to an SL Collection and returns the status of the request (pending or complete). If the status results as pending, the client can periodically poll the status endpoint to get updates on the request processing. A future version of this endpoint will be implemented to enable the sharing of SL Objects as anonymous TAXII Clients.

If the Collection specifies `can_write` as false for a particular client, this endpoint returns an HTTP 403 (Forbidden) error.

Clients can publish duplicates of SL Objects even within the same Collection. Duplicates are considered separate objects and will have two different SL Object id.

<table>
<thead>
<tr>
<th>REQUEST</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td><code>{api-root}/collections/{id}/objects/</code></td>
</tr>
</tbody>
</table>
| URL PARAM | `{api-root}` - Base URL of the SL API Root  
{id} - unique collection identifier |
| HEADER  | Accept: application/taxii+json;version=2.1  
Content-Type: application/taxii+json;version=2.1  
Payload: envelope |
| RESPONSE |  |
| 202     | Request accepted  
Content-Type: application/taxii+json;version=2.1  
Payload: status |
| FAILURE RESPONSES |  |
| 400     | The SL server did not understand the request |
| 401     | The client needs to authenticate |
| 403     | The client does not have access to this collection resource |
| 404     | The SL API Root or Collection ID are not found, or the client cannot write to this object resource |
The media type provided in the Accept header is invalid
413 - The POSTed payload exceeds the max_content_length of the SL API Root
415 - The client attempted to POST a payload with a content type the SL server does not support
422 - The object type or version is not supported or could not be processed. Unprocessable content like malformed body, or not supported versions of STIX
Content-Type: application/taxii+json;version=2.1
Payload: error

III.6.2.d.1. Status endpoint

This endpoint is defined in Table 14. It returns the processing status of a request for a certain status ID. The ID of status is returned with a POST request of an SL Object. Some SL Objects may require more time to be stored in the DR due to network congestion. Clients can monitor the status of a POST request after its submission.

If the SL server receives a status request for a status ID no longer available, the server returns HTTP status of 404 (Not Found).

Table 14 – SL Status API

<table>
<thead>
<tr>
<th>REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /{api-root}/status/{status-id}/</td>
</tr>
<tr>
<td>URL PARAM</td>
</tr>
<tr>
<td>{api-root} - Base URL of the SL API Root</td>
</tr>
<tr>
<td>{status-id} - unique status identifier</td>
</tr>
<tr>
<td>HEADER Accept: application/taxii+json;version=2.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESPONSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 - Successful request</td>
</tr>
<tr>
<td>Content-Type: application/taxii+json;version=2.1</td>
</tr>
<tr>
<td>Payload: status</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FAILURE RESPONSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>401 - The client needs to authenticate</td>
</tr>
<tr>
<td>403 - The client does not have access to this resource</td>
</tr>
<tr>
<td>404 - The SL API Root or Status ID are not found, or the client does not have access to the resource</td>
</tr>
<tr>
<td>406 - The media type provided in the Accept header is invalid</td>
</tr>
<tr>
<td>Content-Type: application/taxii+json;version=2.1</td>
</tr>
<tr>
<td>Payload: error</td>
</tr>
</tbody>
</table>

The status resource type is a JSON containing the status of a POST request to add SL Objects to a collection. The list of objects that failed to be added, are still pending, or have been successfully added is a simple type, named status-details, that contains the identifier of the object (e.g., for STIX objects, their id), its version, and an optional message indicating additional details. The status resource type is defined in Table 15.
### Table 15 – Status resource type

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id (required)</td>
<td>integer</td>
<td>The id property that universally and uniquely identifies this Status.</td>
</tr>
<tr>
<td>status (required)</td>
<td>string</td>
<td>The overall status of a previous POST request where an HTTP 202 (Accept) was returned. The value of this property can be one of complete or pending. Complete indicates that the SL Object has been stored into the DR. Pending indicates that the SL Object is not stored into the DR yet.</td>
</tr>
<tr>
<td>request_timestamp</td>
<td>timestamp</td>
<td>The datetime of the request that this status resource is monitoring.</td>
</tr>
<tr>
<td>total_count (required)</td>
<td>integer</td>
<td>The total number of objects that were in the request, which would be the number of objects in the envelope. It can be equal or greater than zero. If zero, the server has not yet started processing the request.</td>
</tr>
<tr>
<td>success_count (required)</td>
<td>integer</td>
<td>The number of objects that were successfully created.</td>
</tr>
<tr>
<td>successes (optional)</td>
<td>list of status-details</td>
<td>A list of objects that was successfully processed.</td>
</tr>
<tr>
<td>failure_count (required)</td>
<td>integer</td>
<td>The number of SL Objects that failed to be created.</td>
</tr>
<tr>
<td>failures (optional)</td>
<td>list of status-details</td>
<td>A list of SL Objects that was not successfully processed.</td>
</tr>
<tr>
<td>pending_count (required)</td>
<td>integer</td>
<td>The number of pending SL Objects.</td>
</tr>
<tr>
<td>pendants (optional)</td>
<td>list of status-details</td>
<td>A list of pending SL Objects.</td>
</tr>
</tbody>
</table>

The status-details resource type represents an SL Object that was added, is pending, or not added to the Collection. The status-details resource type is defined in Table 16.

### Table 16 – Status-details resource type

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id (required)</td>
<td>string</td>
<td>The id property that universally and uniquely identifies the object that succeed, is pending or failed to be created. It is represented with the unique STIX object id.</td>
</tr>
<tr>
<td>version (required)</td>
<td>string</td>
<td>The version property of the object that succeed, is pending or failed to be created. It corresponds to the STIX object timestamp.</td>
</tr>
<tr>
<td>message (optional)</td>
<td>string</td>
<td>A message indicating information about the object that succeed, is pending or failed to be created.</td>
</tr>
</tbody>
</table>
III.6.2.e. GET an SL Object

This endpoint is defined in Table 17. It gets an SL Object from an SL Collection by its id. The object id corresponds to the STIX id of the object itself. SL Objects are returned as STIX elements wrapped into an envelope resource type (defined in section III.6.2.c.).

If a client fails to authenticate or is not authorised to access this endpoint, it receives an HTTP 401 (Unauthorised) error.

If the Collection specifies can_read as false for a particular client, this endpoint returns an HTTP 403 (Forbidden) error.

Pagination of SL Objects according to certain filtering parameters will be implemented through the validation scenarios.

Table 17 – SL GET Object API

<table>
<thead>
<tr>
<th>REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /{api-root}/collections/{id}/objects/{object-id}</td>
</tr>
</tbody>
</table>

| URL PARAM |
| {api-root} – Base URL of the SL API Root |
| {status-id} – unique status identifier |
| {object-id} – STIX ID of the object being requested |

| URL FILTERING PARAMS – To be designed |
| HEADER Accept: application/taxii+json;version=2.1 |

| RESPONSE |
| 200 – Successful request |
| Content-Type: application/taxii+json;version=2.1 |
| Payload: envelope |

| FAILURE RESPONSES |
| 400 – The SL server did not understand the request |
| 401 – The client needs to authenticate |
| 403 – The client does not have access to this SL Object resource |
| 404 – The SL API Root or SL Collection ID and/or SL Object ID are not found, or the client does not have access to the resource |
| 406 – The media type provided in the Accept header is invalid |
| Content-Type: application/taxii+json;version=2.1 |
| Payload: error |

III.6.2.f. GET SL Object Versions

This endpoint is defined in Table 18. It returns a list of one or more versions on an SL Object in an SL Collection. This list can be used to decide whether it is worth retrieving the actual objects, or if new versions have been added. STIX objects are versioned with their creation timestamp.

If a client fails to authenticate or is not authorised to access this endpoint, it receives an HTTP 401 (Unauthorised) error.

If the Collection specifies can_read as false for a particular client, this endpoint returns an HTTP 403 (Forbidden) error.
Table 18 – SL GET Object Versions API

REQUEST

GET /{api-root}/collections/{id}/objects/{object-id}/versions/

URL PARAM

{api-root} – Base URL of the SL API Root
{status-id} – unique status identifier
{object-id} – STIX ID of the object being requested

URL FILTERING PARAMS – To be designed

HEADER

Accept: application/taxii+json;version=2.1

RESPONSE

200 – Successful request

Content-Type: application/taxii+json;version=2.1

Payload: versions

FAILURE RESPONSES

400 - The SL server did not understand the request
401 - The client needs to authenticate
403 - The client does not have access to this SL Object resource
404 - The SL API Root or SL Collection ID and/or SL Object ID are not found, or the client does not have access to the resource
406 - The media type provided in the Accept header is invalid

Content-Type: application/taxii+json;version=2.1

Payload: error

The versions resource type is a wrapper around a list of versions and is defined in Table 19.

Table 19 – Versions resource type

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>more (optional)</td>
<td>boolean</td>
<td>This property identifies if there is more content available based on the search criteria. The absence of this property means the value is false.</td>
</tr>
<tr>
<td>versions (optional)</td>
<td>list of string</td>
<td>List of SL Objects versions. If no versions available, this parameter is omitted, and the response is an empty object.</td>
</tr>
</tbody>
</table>

III.6.3. Other Interfaces

Implementation Status: design

Beyond the TAXII standard interfaces described above, the SL will expose additional REST APIs for:

- Users management:
  - Create new user;
  - Update user details;
  - Remove a user;
  - Add user to a sharing group;
Manage user encryption keys;

- Interact with the blockchain and DDR:
  - Show the blockchain node status on an SL Instance;
  - Show blockchain accounts on an SL Instance;
  - Retrieve the list of tokenised CTI owned by a blockchain account;
  - Show the DDR node status.

More APIs can be implemented for specific validation scenarios.

Additionally, is under investigation the implementation of a web interface with which users can access and interact with SL carrying manual operations.

### III.6.4. SL Interfaces with other Tools within the CK4SME Toolkit

#### III.6.4.a. SL <> Keenaï

The interface described in this section has been firstly defined in [2] in section IV.5.3. The Keenaï / Service Ledger integration will be based on the use of the TAXII APIs detailed above. The communication between the tools is bi-directional and the exchange of information may be considered in both directions:

- From SL to Keenaï: Keenaï pulls STIX objects from SL (either manually or automatically) to retrieve threat information (e.g. indicators, TTPs, security events) and to update its real-time detection module;
- From Keenaï to SL: Keenaï sends information related to a new or existing threat information to enrich the knowledge base.

### III.6.5. SL Interfaces for Use cases

The use cases proposed by the validation scenarios to date will be supported by the TAXII APIs. The activities on SL so far identified by the use cases make use of the SL API endpoints to:

- Authenticate end users;
- Trigger manual CTI sharing activities;
- Retrieve CTI from sectorial supply chains.

### III.7. Usage

Authenticated users will trigger three actions:

- Shared CTI information with everyone, the authorities or a sectorial sharing group;
- Share CTI as an anonymous account with everyone on the network;
- Retrieve CTI according to its granted permissions.

In addition, SL will offer admin functions to manage users. At this stage of the project, the array of admin functions is still under discussion with SMEs partners defining the validation scenarios.

**Possible usage of SL.** An organisation taking part in the sharing group A and B:

- Retrieves all the shared CTI objects within the sharing group B and updates its SIEM;
- Shares security events exclusively with the sharing group A;
• Periodically retrieves the CTI shared by local authorities on the public sharing group and updates its SIEM.

Shares with the entire community a particular CTI indicator related to malware that affected its infrastructures. In that case, the organisation shares the data with an anonymous transaction to hide its identity avoiding reputation damage.

III.8. Validation

This section describes the features delivered to WP3 for the definition of validation use case scenarios. The validation scenarios themselves are described in D3.2, where each SME partner has a section with the description of each scenario.

SL offers data sharing features while preserving security, data integrity, privacy and business confidentiality. In this section we list the features offered to build specific use cases with the validation scenarios. These SL features are listed below.

• **Secure data sharing**: CTI shared on a distributed data storage (DDS), and secured over the blockchain
  
  o **Tokenisation of CTI**: CTI data will be secured on the blockchain as digital tokens. Any token will point at its relative data source stored on DDS. Tokenised CTI are immutable and tamper-proof representations of shared data. This enables SMEs to share and collect trusted and verifiable data;

• **Privacy preserving sharing**: SMEs can decide to store anonymised data on the DDS to preserve privacy. In this case, the blockchain’s tokenised versions of CTI data will be properly anonymised too;

  o **Tokenisation of CTI**: CTI data will be secured on the blockchain as digital tokens. Any token will point at its relative data source stored on DDS. Tokenised CTI are immutable and tamper-proof representations of shared data. This enables SMEs to share and collect trusted and verifiable data;

• **Custom sharing groups**: SMEs will have the possibility to share data amongst custom groups to preserve business confidentiality. This feature will take advantage of data encryption for DDS (data will be accessible only by group members) and private/permissioned blockchain networks.

III.9. Next Steps

This section provides a detailed roadmap of the SL implementation. This roadmap follows the technical coordination Gantt chart of the project and details the implementation steps that will follow in the next months. The implementation phase of the SL tool will start at the M17 of the project and will proceed till M34.

The roadmap is divided into four main tasks as shown in Figure 30, describing the SL functionalities that will be implemented for the CyberKit4SME project. The tasks and their schedule are so organised:

**M17-M27**: Implementation of the secure data sharing functionalities. In this task, we will implement the TAXII APIs on the SL TAXII Server and their integration with the DR. This task will enable users to share CTI data over a fully decentralised and secure platform.

**M24-M27**: Integration of additional TAXII Collections to enable sharing groups. This task will be characterised by two important phases: (i) the implementation of an SL Collection for each sharing group identified with the use cases; (ii) the creation of access control rules to protect the resources.
M26-M28: Implementation of anonymous transactions. We will go through the realisation of APIs enabling the sharing of fully anonymous CTI – no source will be stored either on the blockchain nor on the DDR.

M28-M34: Further implementations eventually identified with the validation scenarios, including APIs for user management, APIs for querying the blockchain and the DDR, and the implementation of a web interface.

We plan to release the first prototype of SL for validation at the M26 of the project.

---

**SL Roadmap**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Implement TAXII APIs</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Store data on DR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Create Algorand smart-contract for <strong>CTI tokenisation</strong></td>
<td>M17</td>
<td>M27</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Enable sharing groups</td>
<td>M24</td>
<td>M27</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Privacy Preserving Sharing</td>
<td>M26</td>
<td>M28</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Further Implementation:**

- User Management APIs
- Blockchain and DDR APIs
- Other APIs from validation scenarios
- Web Interface

M28 M34 X X X X X X X X X

**First SL Prototype Delivery**

Figure 30 SL Roadmap
IV. CONCLUSIONS

The initial work done on Secure Data Service definition and implementation of the first version is presented, together with its goals, architecture, interfaces and next steps. In addition, the initial work done on the Service Ledger architecture is shown, together with its goals, architecture, interfaces and next steps.

Summary of next steps:

- Expand PME - Dataset integrity protection, PME in PyArrow
- Respond to feedback from Use case partners on the SDS with SL collected in validation iterations
- Work on more ease-of use of the SDS: easier deployment, easier configuration and usage
- Implement the SL architecture based on blockchain and decentralised data storage services for secure data sharing
- Implement the integration between SL and Keenaï
V. REFERENCES


[22] High-level Python API for Parquet encryption of files PR - https://github.com/apache/arrow/pull/10450


[29] Introduction to TAXII - https://oasis-open.github.io/cti-documentation/taxii/intro


[33] The InterPlanetary File System (IPFS) - https://ipfs.io

[34] The Solid Project - https://solidproject.org


[38] Docker - https://www.docker.com


[40] Algorand Sandbox - https://github.com/algorand/sandbox