GUIDE

STORMSHIELD KEY MANAGEMENT AS A SERVICE

ADMINISTRATION GUIDE

Version 4.6

Document last updated: March 31, 2026

Reference: sds-en-sds-kmaas-administration_guide-v4.6

Table of contents

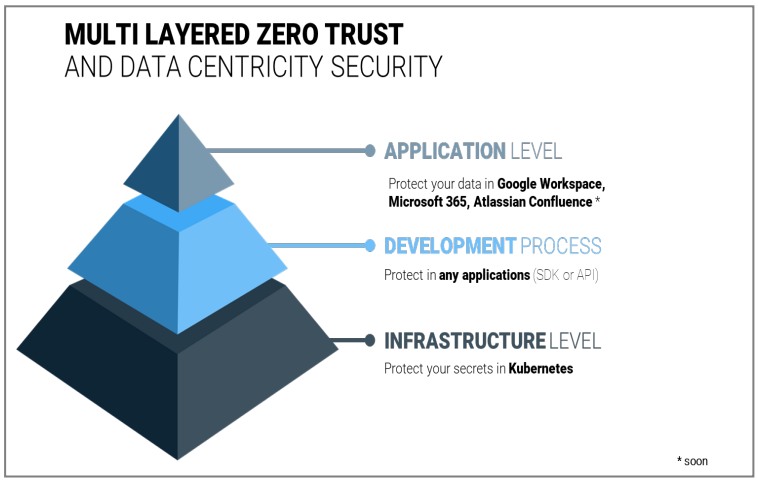
In the documentation, Stormshield Key Management as a service is referred to in its short form: Stormshield KMaaS.

This document is not exhaustive and minor changes may have been included in this version.

1. Getting started

The Stormshield Encryption Platform (SEP) solution helps implementing Data Centric Security and Zero Trust (ZT) in your environment at multiple levels:

- Application level: Integrate Zero Trust directly into your existing applications, or those currently under development (e.g., Google Workspace, healthcare applications, IoT, business),
- Development process: Secure your data and access to your data during your development processes. For instance by securing private HTTPS keys or API tokens for CI/CD, GitOps, etc.,
- Infrastructure level: Protect your secrets at the lowest level in your deployments, especially by securing Kubernetes.

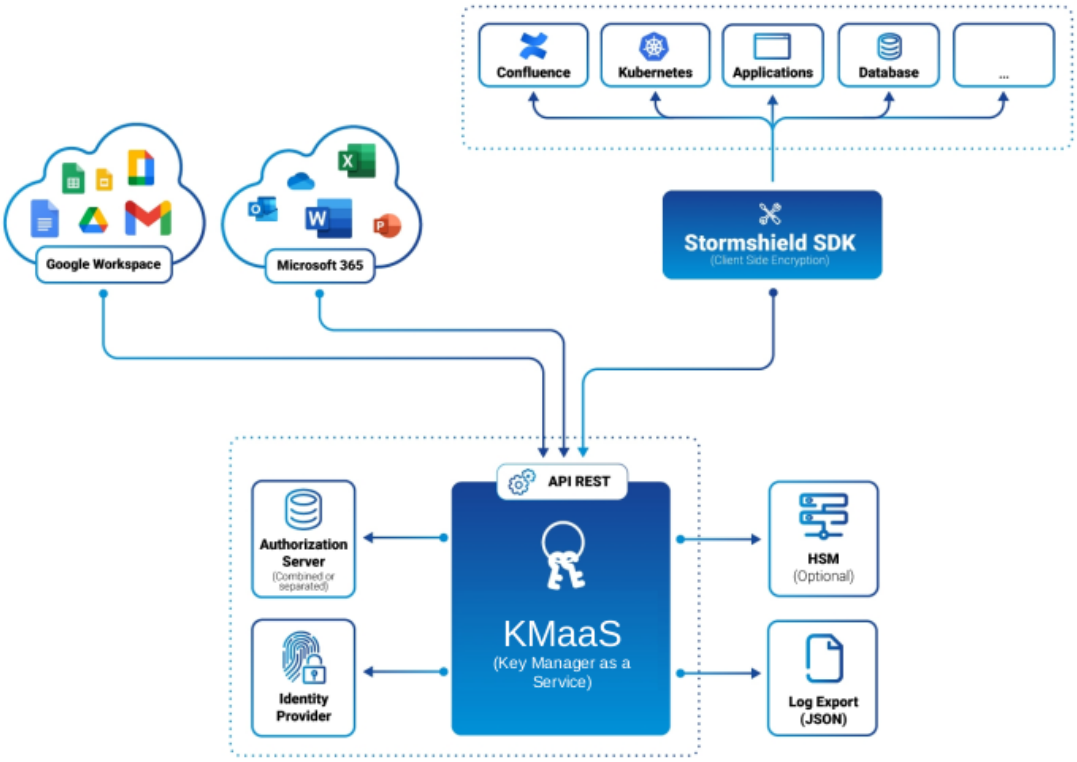


Stormshield KMaaS is the backend component of this ecosystem and acts as a Policy Decision Point as defined in the Zero Trust architecture, securing and authorizing access to confidential data.

It includes the following major modules:

- Key Access Control List Service (KACLS), is dedicated to securing Google Workspace and defined in collaboration with Google,
- Key Access Service (KAS) is a suite of REST APIs used by Stormshield Software Development Kit (SDK),
- Crypto API exposes a REST API for general-purpose cryptographic operations, independent of any specific ecosystem,
- Public Key Infrastructure (PKI) is a service that allows certificates to be quickly issued to secure short-duration MTLS communications.
- Double Key Encryption (DKE) is dedicated to securing Microsoft Office files and emails in a Windows environment.

The diagram below shows the overall Stormshield Encryption Platform (SEP) ecosystem:


Confluence
Kubernetes
Applications
Database
...
Google Workspace
Microsoft 365
Stormshield SDK
(Client Side Encryption)
API REST
Authorization Server
(Combined or separated)
Identity Provider
KMaaS
(Key Manager as a Service)
HSM
(Optional)
Log Export (JSON)

2. Understanding the global requirements

2.1 Requirements

- The server on which the Stormshield KMaaS is installed must be healthy. There must be an information system security policy whose requirements are met on the servers. This policy shall verify the installed software is regularly updated and the system is protected against viruses and spyware or malware (firewall properly configured, antivirus updates, etc.). It is imperative to follow the operating system security recommendations issued by the ANSSI in their document *ANSSII-BP-028-EN*.
- Access to the administrative functions of the workstation system is restricted only to system administrators.
- The operating system must manage the logs generated by the product in accordance with the security policy of the company. It must for example restrict read access to these logs to only those explicitly permitted. For more information, see the section Managing logs.
- You must set up a system upstream of the Stormshield KMaaS to protect against distributed denial-of-service (DDoS) and brute-force attacks. Please follow the ANSSI recommendations (French only).
- You must filter incoming requests upstream of the Stormshield KMaaS. Only requests meeting the following conditions should be accepted:
  - The request header size must be smaller than the NodeJS default value. See the NodeJS documentation.
  - The size of the request body must be less than 1 MB.
- The Stormshield KMaaS must be installed on a server whose system and contributions are kept up to date.
- The server hosting the solution must be located in a secure physical environment with access control protocols and must be trusted.

2.2 Recommendations on administrators

- The Stormshield KMaaS administrators are considered as trusted. They are responsible for defining the Stormshield KMaaS security policy by respecting the state of the art.
- The system administrator responsible is also considered as trusted. He/She is responsible for the installation and maintenance of the application and server. He/She applies the security policy defined by the Stormshield KMaaS administrators.

2.3 Recommendations on network rules

The content of the requests processed by the Stormshield KMaaS is in JSON format, except for the */simpleenroll* route. You can add the following rules to your web firewall (WAF) or load balancer to ensure optimum protection.

- You must block all HTTP requests except:
  - The POST requests with a Content-Type header containing "application/json".
  - The GET and OPTIONS requests without a Content-Type header or with a Content-Type header containing "application/json".

- Be sure to open the following network streams for the modules you use:

Route	Origin wildcard (* = all URLs are allowed)	Allowed method	Module
/health	*	GET	All
/api/v1/<tenant_id>/crypto/decrypt	*	POST	Crypto API
/api/v1/<tenant_id>/crypto/encrypt	*	POST	Crypto API
/api/v1/<tenant_id>/.well-known/est/simpleenroll	*	POST	PKI
/api/v1/<tenant_id>/certs	*	GET	KACLS
/api/v1/<tenant_id>/delegate	.google.com	POST	KACLS
/api/v1/<tenant_id>/digest	.google.com	POST	KACLS
/api/v1/<tenant_id>/status	*	GET	KACLS
/api/v1/<tenant_id>/privatekeydecrypt	.google.com	POST	KACLS
/api/v1/<tenant_id>/privatekeysign	.google.com	POST	KACLS
/api/v1/<tenant_id>/privilegedprivatekeydecrypt	*	POST	KACLS
/api/v1/<tenant_id>/privilegedunwrap	*	POST	KACLS
/api/v1/<tenant_id>/privilegedwrap	*	POST	KACLS
/api/v1/<tenant_id>/rewrap	.google.com	POST	KACLS
/api/v1/<tenant_id>/unwrap	.google.com	POST	KACLS
/api/v1/<tenant_id>/wrap	.google.com	POST	KACLS
/api/v1/<tenant_id>/wrapprivatekey	*	POST	KACLS
/api/v1/<tenant_id>/kas/decrypt	*	POST	KAS
/api/v1/<tenant_id>/kas/encrypt	*	POST	KAS
/api/v1/<tenant_id>/kas/rewrap	*	POST	KAS
/api/v1/{tenantid}/dke/{keyId}	*	GET	DKE
/api/v1/{tenantid}/dke/{keyId}/{versionId}/Decrypt	*	POST	DKE

- On all the routes that you use, activate the OPTIONS method, that enforces the across-origin resource sharing (CORS).

3. Installing and running Stormshield KMaaS

There are two ways to deploy the Stormshield KMaaS: via an RPM for RedHat systems, or via a Docker image. Stormshield recommends using the Docker image whenever possible.

3.1 Installing the Stormshield KMaaS via a Docker image

3.1.1 Requirements

- You must follow the ANSSI's recommendations from the ANSSI-FT-082 document, relating to the deployment of Docker containers.
- You must set up a container orchestration environment (e.g., Kubernetes , Docker Swarm) to automatically manage replication, high availability and container life cycle. For a resilient installation, Stormshield recommends a minimum of 3 instances of the Stormshield KMaaS.
- The configuration of the container orchestrator depends on the technology used. Refer to your orchestrator's documentation for detailed installation steps and security best practices specific to your environment.
- Install Docker on each server where you want to run the Stormshield KMaaS. The minimum Docker version supported is 20.1.1. For more information, refer to the Install Docker Engine documentation.

3.1.2 Knowing the contents of the Docker image artifact

You must contact Stormshield to get the Docker image artifact provided as a compressed folder.

The archive of the Stormshield KMaaS contains the following files:

Location	Resource
stormshield-kmaas-{version}.tar	Docker image of the Stormshield KMaaS in *.tar* format.
config.json.template	Template configuration file for the Stormshield KMaaS.
keks.json.template	Template file for the list of key encryption keys (KEK).
policy.wasm	Default security policy module. This module does not enable any security policies.
policy.data.json	Data file used by the *policy.wasm* module.
application-sbom.xml	Software Bill Of Materials in the standard CycloneDX format.

3.1.3 Checking the artifact integrity

The artifact is provided with its sha256sum, *docker.zip.sha256sum.txt*.

1. Use the following command in the folder containing the artifact:
   `sha256sum --check --status docker.zip.sha256sum.txt`
   It must return `docker.zip : OK`.
2. If the result is different, do not proceed further and contact Stormshield.

3.1.4 Loading Docker

- Load the image of the Stormshield KMaaS in Docker using the following command:

```
docker load --input stormshield-kmaas-<version>.tar
```

1. Create a dedicated directory to host your configuration, in which you copy the template files provided with the Docker image. Rename the files as follows:
   - *config.json*: configuration file of the Stormshield KMaaS,
   - *keks.json*: file containing the list of key encryption keys (KEK).

   In step Configuring the Stormshield KMaaS, you can edit these files directly in this directory.
2. Make sure that the directory containing the configuration files is available in the container through a volume or using your orchestrator's technology.
   For more information, please refer to your orchestrator documentation.
3. Ensure that the *keks.json* file and the private keys are made available in a secure way in the production environment.

3.2 Running Stormshield KMaaS in Docker mode

3.2.1 Requirements

Execution UID/GUID

The Stormshield KMaaS runs with the node user (UID/GID 1000) in the container. To ensure that your application runs correctly and securely, be sure to specify the execution UID/GID correctly. Below is an example of a command:

```
docker run -u 1000:1000 stormshield/kmaas:<version>
```

File access

- You must allow containers to access your configuration files (i.e., *keks.json*, *config.json*, OPA files, certificate files and private keys).
- For *config.json* and *keks.json* files, Stormshield makes the following recommendations:
  - If you have several instances of the Stormshield KMaaS, expose a single file to the various containers of the application, as they must be identical on all instances,
  - Mount them read-only, as they will never be modified by the Stormshield KMaaS.
    Below is an example of a command with a read-only folder containing configuration files:
    ```
    docker run -v -u 1000:1000 /my-kmaas-config-
    folder:/etc/stormshield/cse:ro stormshield/kmaas:<version>
    ```
- Sensitive files (i.e., *keks.json*, private keys) must be managed by secure mechanisms provided by your orchestrator. Refer to the documentation of your orchestrator.

Network traffic redirection

The service listens on the port defined in the *config.json* file (3000 by default) in the container. Below is an example of a command that forwards host port 443 to port 3000:

```
docker run -p 443:3000 my-image
```

Refer to your orchestrator's documentation to set up port forwarding in a production environment.

Access to environment variables

Containers must have access to the environment variables mentioned in this administration guide. Below is the command for declaring an environment variable:

```
docker run -e MY_VARIABLE=my-variable-value
stormshield/kmaas:<version>
```

Refer to your orchestrator's documentation to set the environment variables in a production environment.

3.2.2 Starting a container

Example of a Docker command to start a Stormshield KMaaS container:

```
docker run -v /my-kmaas-config-folder:/etc/stormshield/cse:ro
-p 443:3000 -u 1000:1000 stormshield/kmaas:4.6.0.268
```

3.3 Installing the Stormshield KMaaS via RPM

Before installing the Stormshield KMaaS, you must install the operating system and NodeJS.

3.3.1 Requirements

Stormshield recommends that the server on which the Stormshield KMaaS is installed has a multi-core processor with a minimum of 4 cores.
In a cluster of three servers for the Stormshield KMaaS, in order to manage an average of 45 requests per second and per Red Hat instance, each server must have at least the following resources:

- 4 processors and one thread per processor
- 4 GB of memory
- 20 GB of storage

If you want to improve performance, add the following resources in this order:

1. Threads for each processor,
2. Processors,
3. Instances in the cluster.

3.3.2 Compatibility

Each supported version of the operating system is compatible with specific versions of OpenSSL and NodeJS. Please check the compatibility in the table below:

Operating system	OpenSSL version	NodeJS version
RedHat Enterprise Linux 8.10	At least v3.2.X	v20 Tested with v20.16.0
RedHat Enterprise Linux 9.6	At least v3.2.2	v22

3.3.3 Installing the operating system

Install and activate a RedHat Enterprise Linux distribution version 8.10 or 9.6 based on the version of the RPM delivered by Stormshield.

For more information, refer to the RedHat 8 documentation or the RedHat 9 documentation.

It is imperative to follow the operating system security recommendations issued by the ANSSI in their document *ANSSII-BP-028-EN*.

You can install all dependencies offline on your operating system. To do so:

1. Get the RPM of the dependency.
2. Copy it on your machine.
3. Install it by running the command:
   `rpm -i`

3.3.4 Installing OpenSSL

- OpenSSL v1.1.1 is supplied by default with RedHat Enterprise Linux 8.10. You must manually install OpenSSL 3 using the commands below. Stormshield recommends using the EPEL repository.

```
# subscription-manager repos --enable codeready-builder-for-rhel-8-
$(arch)-rpms

# dnf install https://dl.fedoraproject.org/pub/epel/epel-release-
latest-8.noarch.rpm

# dnf install openssl3

# ln -b -s /usr/bin/openssl3 /usr/bin/openssl

# ln -b -s /usr/lib64/libssl.so.<openssl_version>
/usr/lib64/libssl.so

# ln -b -s /usr/lib64/libcrypto.so.<openssl_version>
/usr/lib64/libcrypto.so

# ln -b -s /usr/include/openssl3/openssl /usr/include/openssl
```

  where <openssl_version> must be replaced by the OpenSSL version installed, for example *3.2.2*.
- OpenSSL v3.2.X is supplied by default with RedHat Enterprise Linux 9.0. Install OpenSSL using the following command:
  `# yum install openssl`

3.3.5 Installing NodeJS

1. Install the package using the following commands:
   - For RedHat 8:
     `# curl -fsSL https://rpm.nodesource.com/setup_20.x | bash -# dnf install -y nodejs`
   - For RedHat 9:
     `# dnf module install nodejs:22`
2. Check that NodeJS has indeed been installed with the following command:
   `# node --version`

Ensure that the NodeJS autorun has been enabled.

This command will install the latest version of NodeJS 20. The version 4.6 of the Stormshield KMaaS has been tested with NodeJS 20.16.0.

With ReHat 9, if Node.js is installed as instructed above, it uses a dynamic link with the *libcrypto* and *libssl* libraries of OpenSSL. It automatically gets the latest security patches provided by RedHat.

3.3.6 Installing the Stormshield KMaaS

The RPM is provided as an archive with:

- The Software Bills of Materials file (i.e. *application-sbom.xml*) in the CycloneDX format,
- Its sha256sum to check its integrity (i.e. *rpm_redhatx.sha256sum.txt*).

To check the RPM integrity:

1. Use the following command in the folder containing the RPM:
   `sha256sum --check --status rpm_redhatx.sha256sum.txt`
   It must return `rpm_redhatx.zip : OK`.
2. If the result is different, do not proceed further and contact Stormshield.

To install the Stormshield KMaaS, you must be a root user of the RedHat system.

1. Copy the `cse-4.6.x.xxx-redhatx.x86_64.rpm` file on the system.
2. Run the following command:
   `# rpm -i <package_name>.rpm`

If NodeJS was not installed beforehand, this error message will appear:

```
error: Failed dependencies : nodejs is needed by csexxx
```

The following folders and files will be installed:

Location	Resource
/usr/lib64/cse	Source file folder. On installation, the owner of the files is the user *stormshield-cse.* He has u=rx,g=,o= permissions. For security reasons, we recommend keeping these default settings.
/usr/bin/cse	Binary file folder
/etc/stormshield/cse	Configuration file folder: • *config.json.template* - template configuration file for the Stormshield KMaaS. • *keks.json.template* - template file for the list of key encryption keys (KEK). • *policy.wasm* - default security policy module. This module does not enable any security policies. • *policy.data.json* - data file used by the policy.wasm module.
/etc/systemd/system/cse.service	Configuration file to use the Stormshield KMaaS as a SystemD service
/usr/share/licenses/cse	License file folder
/usr/share/doc/stormshield/cse/copyright	Folder of the license files for the open-source libraries

3.4 Running Stormshield KMaaS in RPM mode

1. Run the Stormshield KMaaS as a systemd service with the root user using the following command:
   ```
   # systemctl start cse
   ```
2. Check the status of the service:
   ```
   # systemctl status cse
   ```
3. Enable the autorun of the service when the machine starts:
   ```
   # systemctl enable cse
   ```

The Stormshield KMaaS uses all available CPU cores. This parameter cannot be configured.

3.5 Checking system health

After you have installed and run the Stormshield KMaaS, check that it is running correctly.

This API returns only a limited amount of information about the running of the Stormshield KMaaS and does not require any CORS in the HTTP header.

1. Use the *health* API route:
   ```
   curl https://<my-cse-url>/health
   ```
2. If the system is running correctly, the return must be in the following form:
   ```
   {}
   ```

4. Uninstalling the Stormshield KMaaS

4.1 In Docker mode

1. Clean up the containers:
   ```
   docker stop <container_name>
   docker rm <container_name>
   ```
2. Delete the local image:
   ```
   docker rmi stormshield/kmaas:<version>
   ```

4.2 In RPM mode

1. Run the following command as a user with administration privileges:
   ```
   # rpm -e cse
   ```
   The files added when installing the RPM are deleted, except:
   - The files that you have modified in the meantime. They are saved with the *.rpmsave* extension.
   - The file that you have added yourself. They are kept.
2. Manually delete the configuration files that you have created or modified: *config.json*, *keks.json, policy.wasm and policy.data.json.*

> **NOTE**
> KEKs are highly sensitive items in terms of security. It is imperative to follow the ANSSI recommendations concerning their life cycle.

5. Configuring the Stormshield KMaaS

The global configuration of the Stormshield KMaaS is managed through a JSON file, *config.json*, which is saved by default in */etc/stormshield/cse*. This file sets the specifications for authentication and authorization, as well as the port, service name and service mode.

5.1 Creating the global configuration file

A template file, *config.json.template*, is available to assist you.

- In RPM mode, this file is located in the directory */etc/stormshield/cse*.
  - Create your own global configuration file from the copy of the template using the following command:
    ```
    # cd /etc/stormshield/cse
    # cp --preserve config.json.template config.json
    ```
- In Docker mode, the file is located in the dedicated directory you have created during installation. For more information, see Installing the Stormshield KMaaS via a Docker image.

5.2 Assigning access privileges to the file

- Assign the read and write access privileges held by the current user to the file and read access to the current *config.json* group:
  ```
  # chmod u=rw,g=r,o= config.json
  ```

Do not assign any run privileges on these files, or any privileges to other users. If access privileges are too permissive, a warning log will be generated when the Stormshield KMaaS starts, but will not prevent it from launching.

During installation, the *stormshield-cse* user is the owner of the configuration files by default. Do not change the owner.

5.3 Editing the global configuration file

- To edit the configuration, change the default values of the *config.json* file template.

For more information on this file, refer to sections Knowing the contents of the Docker image artifact and Installing the Stormshield KMaaS

The tables below describe the parameters in the *config.json* file. The first table lists simple parameters which do not contain any sub-objects. More complex parameters have their own table.

Unless otherwise specified, in the configuration files:

- All “String” fields are restricted to 10,000 characters for security reasons,
- All ”Array” fields are limited to 500 items for security reasons.

5.3.1 Simple parameters

Parameter	Description	Type	Optional/ mandatory
kacls_url	The Stormshield KMaaS URL used to prevent "Man-in-the-middle" attacks. E.g., *https://<cse.example.com>*.	String	Mandatory
port	Port on which the Stormshield KMaaS listens. Port 3000 by default.	Integer	Optional
name	Name of the Stormshield KMaaS.	String	Optional
persistence_type	Mode used for storing KEKs. The prescribed values are: • "json_file" if keys are stored in the *keks.json* file, • "kms"' if keys are stored in a key management system (KMS). • "database" if keys are stored in a PostgreSQL database.	String	Mandatory
kacls_kek_label	Label used to identify and retrieve the KEKs from the KMS, if the KEKs are stored in a KMS.	String between 1 and 255 characters long.	Mandatory if "persistence_type": "kms"
external_request_timeout	Timeout in milliseconds before canceling an external request (7000 ms by default). This timeout does not apply to the KMIP-related requests when using a KMS.	Integer	Optional
jwt_supported_signing_algorithms	List of the allowed signature algorithms for checking the validity of authorization and authentication tokens. Supported algorithms are: ["RS256", "RS384", "RS512", "ES256", "ES384", "ES512", "PS256", "PS384", "PS512"].	String	Mandatory (at least one algorithm)

5.3.2 tenants parameter

Contains configuration information specific to each tenant. They are grouped by "tenant_id", which is the tenant unique identifier and is mandatory. The "tenants" object includes the following components:

Parameter	Description	Type	Optional/ mandatory
tenant_id	Unique identifier of the tenant in UUIDv4 format.	String	Mandatory
kacls: JSON object containing the configuration for the KACLS module. For more information, refer to Configuring the KACLS.			
enable	Enables or disables the KACLS module.	Boolean	Mandatory

Parameter	Description	Type	Optional/ mandatory
crypto_api: JSON object containing the configuration for the Crypto API module. For more information, refer to Configuring Crypto API.			
enable	Enables or disables the Crypto API module.	Boolean	Mandatory
pki: JSON object containing the configuration for the PKI (Public Key Infrastructure) module. For more information, refer to Configuring the PKI.			
enable	Enables or disables the PKI module.	Boolean	Mandatory
kas: Json object containing the configuration for the Key Access Service (KAS) module. For more information, refer to Configuring the Key Access Service.			
enable	Enables or disables the KAS module.	Boolean	Mandatory
dke: Json object containing the configuration for the Double Key Encryption (DKE) module. For more information, refer to Configuring the DKE.			
enable	Enables or disables the DKE module.	Boolean	Mandatory
admin: Json object containing the configuration for the Admin module. For more information, refer to Configuring the Admin module.			
enable	Enables or disables the Admin module.	Boolean	Mandatory

5.3.3 authorization parameters

Array of JSON objects that describes how the KACLS authorization token generated by Google is verified. It includes the following components.

Add one entry per Google service (e.g., Meet, Drive, Calendar, Gmail). The values of these settings are provided in the Google documentation. For more information, see Example of the authorization parameter for Google services

The authorization parameter is mandatory in all configurations. When using the KACLS, it must contain the required authorization data. If you do not use the KACLS, the parameter must be an empty array (*"authorization": []*).

Parameter	Description	Type	Optional/ mandatory
issuer	Issuer of the JWT authorization token (see RFC 7519).	String	Optional
url	URL to the JWKS JSON file.	String	Mandatory
audience	Recipient of the JWT authorization token (see RFC 7519).	String	Optional

Example of the authorization parameter for Google services

This extract of the *config.json* file is an example of how the `authorization` token can be configured for the Drive, Meet, Calendar and Gmail Google services, the migration of one KACLS to another, and Gmail Send to Anyone option. You can customize the rules that allow or deny a

request to the Stormshield KMaaS, using Open Policy Agent (OPA) policies. For more information, see the section Implementing the authorization rules with Open Policy Agent.

```
"authorization": [
    {
    "url": "https://www.googleapis.com/service_accounts/v1/jwk/gsuitecse-tokenissuer-
drive@system.gserviceaccount.com",
    "issuer": "gsuitecse-tokenissuer-drive@system.gserviceaccount.com",
    "audience": "cse-authorization"
    },
    {
    "url": "https://www.googleapis.com/service_accounts/v1/jwk/gsuitecse-tokenissuer-
meet@system.gserviceaccount.com",
    "issuer": "gsuitecse-tokenissuer-meet@system.gserviceaccount.com",
    "audience": "cse-authorization"
    }
    {
    "url": "https://www.googleapis.com/service_accounts/v1/jwk/gsuitecse-tokenissuer-
calendar@system.gserviceaccount.com",
    "issuer": "gsuitecse-tokenissuer-calendar@system.gserviceaccount.com",
    "audience": "cse-authorization"
    }
    {
    "url": "https://www.googleapis.com/service_accounts/v1/jwk/gsuitecse-tokenissuer-
gmail@system.gserviceaccount.com",
    "issuer": "gsuitecse-tokenissuer-gmail@system.gserviceaccount.com",
    "audience": "cse-authorization"
    }
    {
    "url": "https://www.googleapis.com/service_accounts/v1/jwk/apps-security-cse-
kaclscommunication@system.gserviceaccount.com",
    "issuer": "apps-security-cse-kaclscommunication@system.gserviceaccount.com"",
    "audience": "cse-authorization"
    }
    {
    "url": "https://www.googleapis.com/service_accounts/v1/jwk/gsuitecse-tokenissuer-
gmail-sta@system.gserviceaccount.com",
    "issuer": "gsuitecse-tokenissuer-gmail-sta@system.gserviceaccount.com",
    "audience": "cse-authorization"
    },
    ]
```

5.3.4 https parameter

JSON object that describes the HTTPS certificates. To be used when you want to run the server in secure mode. It includes the following components:

Parameter	Description	Type	Optional/ mandatory
credentials	• key: path to the private HTTPS key in PEM format. • cert: path to the HTTPS certificate in PEM format. • ca (optional): path to the HTTPS certification authority in PEM format.	Object	Optional

5.3.5 keks parameter

JSON object describing the refresh frequency of the KEKs while the Stormshield KMaaS is running. It includes the following components:

Parameter	Description	Type	Optional/ mandatory
auto_refresh	• scheduled.interval_seconds: frequency at which the KEKs are scheduled to be automatically refreshed by the Stormshield KMaaS (by default 86400 seconds, minimum value 1800 seconds). Make sure you specify a value matching your needs and use. Any value lower than 1800 seconds (30 minutes) will be considered invalid and will prevent the server from starting. • minimum_interval_seconds: minimum interval between two refresh operations, whether periodic or one-off (3600 seconds by default). Limits queries to the KMS. KEK refresh is only applicable if the parameter persistence_type: "kms"	Integer in seconds	Optional

5.3.6 kmip_configuration parameters

JSON object that describes the KMS configuration. It is mandatory if "persistence_type": "kms". It includes the following objects:

Parameter	Description	Type	Optional/ mandatory
host	KMS server.	String	Mandatory
port	Port that the KMS listens on (optional, 5696 by default).	Integer	Optional
client_private_key_path	Path to the private key of the KMS client in PEM format.	String	Mandatory
client_certificate_path	Path to the certificate of the KMS client in PEM format	String	Mandatory
ca_certificate_path	Path to the certification authority of the KMS client in PEM format.	String	Optional

In order to use a KMS domain, the certificate of the KMS client must have a common name that conforms with the *KMS_DOMAIN_NAME*>||<*KMS_USER_NAME*> format.
For instance, if your user is *Jane* and your domain is *JaneDomain*, the common name must be: *JaneDomain*||*Jane*.
For more information, refer to the KMS KMIP documentation.

If using a KMs domain, the *domain_id* values of all the tenant configurations must imperatively be the same and match the domain used by the KMIP configuration. For more information, refer to section crypto_backends.

WARNING
Stormshield does not guarantee the proper functioning of the product if you do not follow these instructions.

NOTE:
The maximum number of simultaneous connections to the KMS server depends on your infrastructure and the KMS configuration.

5.3.7 cache parameter

JSON object that describes the configuration of the cache. Set the values according to your configuration, e.g., available memory. It includes the following objects:

Parameter	Description	Type	Optional/ mandatory
enable	Activation status of the cache (true by default).	Boolean	Optional
max_cache_capacity	Maximum capacity of the cache (100 MB by default).	Integer in MB	Optional
max_object_size	Maximum size of a cached object in KB (100 KB by default). OpenId, jwks and remote configurations (*cse-config*) are cached. Set a value high enough to store these objects.	Integer in MB	Optional
max_object_lifetime	Lifetime of a cached object (1440 min by default). We advise against exceeding the lifetime of tokens provided by the identity providers.	Integer in minutes	Optional

5.3.8 database parameter

JSON object that describes the configuration of the database. Set the values according to your database. For more information, refer to section Requirements.
It includes the following objects:

Parameter	Description	Type	Optional/ mandatory
enable	Activation status of the database.	Boolean	Mandatory
host	Database server host.	String	Mandatory if enable is set to true
port	Database server port.	Integer	Mandatory if enable is set to true
name	Database name.	String	Mandatory if enable is set to true
schema	Schema name. For more information, refer to section Creating the database schema	String	Mandatory if enable is set to true

Parameter	Description	Type	Optional/ mandatory
username	Name of the database user.	String	Mandatory if enable is set to true
authentication: JSON object describing the parameters required to authenticate to the database.			
mode	Authentication mode. Possiblevalues: • *basic* • *mtls*	String	Mandatory if enable is set to true
credentials	• In *basic* mode: • user_password: database password. In *mtls* mode: • ca_certificate_path: path to the certificates of the certificate authority, used to verify the server certificate, • client_certificate_path: path to the certificate of the database client, • client_private_key_path: path to the private key of the database client.	String	Mandatory if enable is set to true

To configure Postgres to use the mTLS mode:

1. Create a key and a certificate for the Postgres server.
   You must have a CA capable of verifying the client certificate.
   For information, refer to the Postgres documentation.
2. Create a key and certificate for the database client (i.e. the KMaaS).
   You must have a CA capable of verifying the server certificate.
   The “Common Name” (CN) field of the client certificate must contain the user name to be used for authentication.
   For information, refer to the Postgres documentation.
3. Fill out the *pg_hba.conf* Postgres file, required to configure the connection.
   For information, refer to the Postgres documentation.

5.3.9 logs parameters

JSON object that allows configuring how to display the logs in the new format. If the object is missing in the *config.json* file, only the logs with the old format are displayed. It includes the following objects.

For more information, see the section Managing logs.

Parameter	Description	Type	Optional/ mandatory
enable	Enable log display.	Boolean	Mandatory

Parameter	Description	Type	Optional/ mandatory
formats	Version of the log format to enable. The prescribed values are: • ["v1", "v2"] if all logs are displayed, • ["v2"] if only logs with the new format are displayed, • ["v1"] if only logs with the old format are displayed, • [] if no logs are displayed.	String list	Mandatory if enable is set to true
kinds	Log family to which the log belongs. Prescribed values: • *domain*: Stormshield KMaaS business operation logs. • *system*: Logs relating to the operations concerning the environment. • *http*: Logs relating to the HTTP operations of the Stormshield KMaaS.	String list	Mandatory if format is set to v2
severities	Level of severity of the log. Prescribed values: • *emerg*: The system is unusable, • *alert*: The problem must be fixed immediately, • *crit*: Critical error, • *err*: Non-critical error, • *warning*: The operation was successful but generated a warning, • *notice*: Unusual event not requiring corrective action, • *info*: Normal operation information message, • *debug*: Information useful to developers for troubleshooting the application.	String list	Mandatory if format is set to v2

6. Configuring KEKs

The Stormshield KMaaS uses key encryption keys, i.e., KEKs, to wrap and unwrap Data Encryption Keys (DEKs). The different services using the Stormshield KMaaS API (i.e., KACLS, Stormshield SDK) provide the DEKs to encrypt/decrypt data.

The KEKs are used by all the modules of Stormshield KMaaS.

There are three ways to store keys:

- Database mode: KEKs are stored in a PostgreSQL database dedicated to the Stormshield KMaaS. This mode provides a good level of security and does not require to implement a third party KMS. It is possible to add KEKs through the Admin module and they are loaded on demand. This mode requires the use of master KEKs (i.e., MKEKs).
- File mode: KEKs are stored in plaintext in the */etc/stormshield/cse/keks.json* file on the server. Access to keys depends on how secure the server is.
- KMS mode: KEKs are stored in a Key Management System (KMS).
  The keys are selected in the KMS by using the value of the *kacls_kek_label* parameter in the *config.json* file. They are refreshed regularly, based on the value of the *keks* parameter in the *config.json* file. See keks parameter.

KEKs are highly sensitive items in terms of security. You must follow the ANSSI recommendations concerning their life cycle.

6.1 Configuring KEKs in standalone mode

6.1.1 Generating KEKs

You can configure several kinds of keys in the Stormshield KMaaS :

- KEKs and Master encryption keys (i.e., MKEK) for symmetric encryption use cases,
- KEKS for asymmetric encryption use cases with the Key Access Service,
- KEKs for Microsoft Double Key Encryption.

The Stormshield KMaaS does not generate KEKs, so you must create them beforehand.

Generating KEKs and MKEKs for symmetric encryption

KEKs for symmetric encryption are 256-byte AES-256 keys listed in the *keks.json* file in the form of base64-encoded character strings.

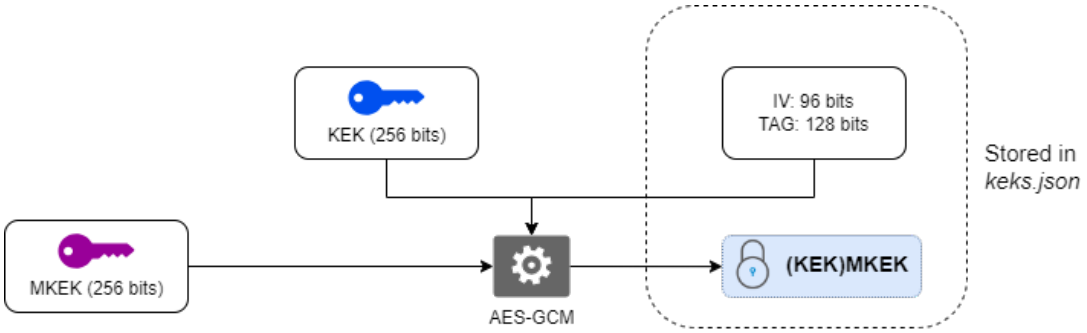
To create KEKs, you can use OpenSSL for example:

- On a Red Hat system, install OpenSSL by using the following command:
  `yum install openssl`.
- On a Linux system, install OpenSSL using the package manager corresponding to your distribution.
- With the `rand` command, data directly encoded in base64 can be generated:
  `openssl rand -base64 32`
- To improve the security of the Stormshield KMaaS, you can optionally use OpenSSL to generate Master encryption keys (i.e., MKEK). An MKEK is used to encrypt your KEKs so that they are not exposed in clear text in the configuration file.

The MKEKs must be as follows:

- They must be 256 bits in size,
- They must be base64-encoded,
- They must encrypt the KEKs with a 12 byte initialization vector and a 16 byte authentication tag.

The diagram below illustrates how the MKEK encrypts KEKs.



For more information on aes-256-gcm encryption to generate the MKEK, the encrypted KEK, the initialization vector and the tag, see the ANSSI document ANSSI-PG-083.

If you are using at least one KEK encrypted by an MKEK, you must declare the MKEK_VALUE environment variable containing the value of the MKEK encryption key before starting the Stormshield KMaaS.

Generating asymmetric KEKs for KAS rewrap

The asymmetric KEKs are only used for wrapping and unwrapping operations via the *kas/rewrap* route. They must be as follows:

- 4096-bit RSA key pairs,
- Each key pair consists of a private and a public key,
- The public key OID must be *1.2.840.113549.1.1.1*.

They are listed in the *keys* section of the *keks.json* file. For more information, refer to section Adding KEKs to the file.

The procedures for managing and securing the files are the same as for symmetric keys. For more information, refer to section Generating KEKs and MKEKs for symmetric encryption.

To create an asymmetric RSA key pair with OpenSSL:

1. Create the 4096-bit private key:
```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:4096
```
2. Extract the public key:
```
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

Generating DKE key pairs

Double Key Encryption (DKE) relies on asymmetric RSA key pairs used by Microsoft to protect Data Encryption Keys (DEKs). Each key pair consists of 2048-bit RSA private and public keys.

They are listed in the *keys* section of the *keks.json* file. For more information, refer to section Adding KEKs to the file.

To generate the keys required for the DKE module, run the following commands:

1. Create the 2048-bit private key:
```
openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out private.pem
```

2. Extract the public key:
```
openssl rsa -in private.pem -pubout -out public.pem
```

6.1.2 Preparing the key encryption key file

The Stormshield KMaaS manages key encryption keys, i.e., KEKs, and stores them in the *keks.json* file. This file must be saved in the directory */etc/stormshield/cse*.

Creating the KEK file

In RPM mode, a file template can be found in */etc/stormshield/cse* to assist you.

- Create your own KEK file from the copy of the template using the following command:
```
# cd /etc/stormshield/cse
# cp --preserve keks.json.template keks.json
```

In Docker mode, the file is located in the dedicated directory you have created during installation. For more information, see Installing the Stormshield KMaaS via a Docker image.

Assigning access privileges to the file

1. Assign the read and write access privileges held by the current user to the *config.json* file and read access to the current group:
```
# chmod u=rw,g=r,o= keks.json
```
2. Set *stormshield-cse* as the file owner.
```
# chown stormshield-cse keks.json
```

If access privileges are too permissive, a warning log will be generated when the Stormshield KMaaS starts, but will not prevent it from launching.

6.1.3 Adding KEKs to the file

After you have generated your KEKs, encrypted or not, add them manually to the *keks.json* file.

The same file can include all types of KEKs: encrypted/not encrypted and symmetric/asymmetric.

Parameter	Description	Optional/ mandatory
tenant_id	UUID v4 of your tenant, the same that you have specified for the External key service.	Mandatory
active_kek_id	ID of the active KEK that will be used to encrypt keys for symmetric encryption.	Mandatory
keks: JSON object array containing the definition of non encrypted KEKs for symmetric encryption.		
id	Unique ID generated in UUID v4 format.	Mandatory
kek_b64	Value of the KEK.	
encrypted_keks: JSON object array containing the definition of the MKEK-encrypted KEKs for symmetric encryption.		
id	Unique ID generated in UUID v4 format.	Mandatory
encrypted_kek_b64	Value of the encrypted KEK. It must imperatively be generated using the *aes-256-gcm* encryption algorithm.	

Parameter	Description	Optional/ mandatory
crypto_material	• crypto_context: object containing the following fields, generated while encrypting the KEK with the MKEK: - iv: initialization vector - tag: authentication tag	
	• encryption_algorithm: must be *aes-256-gcm*	
	• m_kek_location: information about the location of the MKEK used to encrypt the KEK - key_name: name of the environment variable containing the MKEK value - type: must be *env*	
keys: JSON object array containing the definition of the asymmetric KEKs used for KAS rewrap.		
kid	Unique ID generated in UUID v4 format.	Mandatory
private	Value of the private key (PEM format in base64).	Mandatory
public	Value of the public key (PEM format in base64).	Mandatory
active	Indicates if the key is currently enabled.	Mandatory
dke_keys: JSON object array containing the definition of the asymmetric KEKs used for the DKE module. Note that the server does not check at startup if the public key matches the private key.		
kid	Unique ID generated in UUID v4 format.	Mandatory
label	Key display name.	Mandatory
created_at	Date the key was created.	Mandatory
versions	Object array containing information about the keys: • version_id: Version identifier (UUID), • private: Value of the private key in PEM format, base64 encoded, PKCS#8 type, RSA 2048, • public: Value of the public key in PEM format, base64, RSA 2048, • created_at: timestamp (number)	Mandatory

Generate v4 UUIDs with any tools of your choice (e.g., UUID Generator).

EXAMPLES
The file contents below are given simply as examples and must not be used as such in your Stormshield KMaaS configuration.

```
{
   "tenants": [
      {
         "tenant_id": "3a5f06fe-bee2-444b-bf76-b5ead30327c0",
         "active_kek_id": "fd7e4c16-6199-40a3-9bce-3c82a9e31e66",
```

```
            "keks": [
            {
               "id": "fd7e2c15-6199-40a3-9bce-3c82a9e31e66",
               "kek_b64": "9i6NnOFeABODElB+ujySsqK74PPVlW6dhy6mvQt+RaQ="
            }
         ],
         "encrypted_keks": [],
         "keys": [
         {
            "kid": "96b2f83d-df4e-4d49-b662-bcde91a8764f",
            "private":
"LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQ0KTUlJSktBSUJBQUtDQWd[...]
            "public":
"LS0tLS1CRUdJTiBSU0EgUFVCTElDIEtFWS0tLS0tCk1JSUNDZ0tDQWdFQXlsWDNE[...]
            "active": true
         }
         ],
         dke_keys : [{
           created_at: 1763975729,
           kid: 'df77f9d6-052a-4a01-9228-7ad1883d4a50',
           label: 'dke keys',
           status: JsonDkeKeyStatus.Enabled,
           versions: [
             {
              created_at: 1763975729,
              version_id: 'f6bec63f-c5a3-4932-94da-d6a34eb69066',
              private: 'LS0tLS1CRUdJTiBSU0EgU...',
              public: 'LS0tLS1CRUdJTiBQVUJMS...',
             },
          ]}
         ]
      }
      ]
}
```


6.1.4 Renewing a symmetric encryption KEK

When renewing a KEK, previous keys must remain accessible for unwrapping, and the new KEK must not be used until it has been deployed on all Stormshield KMaaS servers.

1. Add a new KEK to the *keks.json* file:
   a. Retrieve the *keks.json* file on one of the servers.
   b. Generate a new AES 256 KEK.
   c. Add this KEK to the file and assign a new unique ID to it.
2. Publish the *keks.json* file successively on each server:
   a. Replace the *keks.json* file with the one modified in step 1. Ensure that you keep the same access privileges as for the existing file.
   b. Run the restart command on the server.
   The server will restart and reload its list of KEKs from the *keks.json* file. The active KEK does not change for the moment.
3. Set the new active KEK in the *keks.json* file:
   a. Edit the *keks.json* file again.
   b. Change the value associated with `active_kek_id` so that it points to the ID of the KEK generated in step 1.

4. Publish the *keks.json* file again on all the servers.
   On each successive server:
   a. Replace the *keks.json* file with the one modified in step 3. Ensure that you keep the same access privileges as for the existing file.
   b. Run the restart command on the server.
   The server will restart and reload its list of KEKs from the *keks.json* file. The active KEK is changed and the server is ready to wrap keys with the new KEK.

   In step 4, if a wrapping request is submitted on a server that uses the new KEK, all the other servers can respond to an unwrapping request regardless of their status, since they all know the new key.

6.2 Configuring encryption KEKs in database mode

6.2.1 Requirements

To use the Stormshield KMaaS with KEKs stored in a database, you must meet the following requirements:

- A PostgreSQL database version 16.x must be available to the Stormshield KMaaS in your infrastructure, with the database schema described in appendix Creating the database schema.
- In the *config.json* file, the following parameters must be set:
  - The "persistence_type" parameter must be set to "database". For more information, refer to section Simple parameters,
  - the "database" parameters must be filled in according to your database configuration. Refer to section database parameter.

6.2.2 Generating KEKs and MKEKs

You can configure two kinds of keys in the Stormshield KMaaS:

- KEKs and Master encryption keys (i.e., MKEKs) for symmetric encryption use cases,
- KEKS for asymmetric encryption use cases with the Key Access Service.

The Stormshield KMaaS does not automatically generate keys, so you must create them beforehand. For more information on requirements and how to create keys, refer to:

- Generating KEKs and MKEKs for symmetric encryption,
- Generating asymmetric KEKs for KAS rewrap .

6.2.3 Adding KEKs and MKEKs to a tenant in the database

The following procedure and SQL scripts describe how to add a key in a configuration with one tenant and only the KACLS module enabled. You must customize the procedure and scripts to match your configuration.

1. Connect to the database dedicated to key storage, and run the following SQL scripts.
2. Create the tenant:

```
# INSERT INTO tenant (tenant_id, display_name)
  VALUES ('<TENANT_ID>', '<OPTIONAL_TENANT_NAME>');
```

3. Associate the module to the tenant. The values are "1" for the KACLS, "2" for the KAS, and "3" for Crypto API.

```
# INSERT INTO module (tenant_id, module_id)
  VALUES ('<TENANT_ID>', 1);
```

4. Add an MKEK to the tenant. The <MKEK_BASE64_ENV_VARIABLE_NAME> refers to the name of the environment variable hosting the MKEK value in base 64. For more information, refer to Generating KEKs and MKEKs for symmetric encryption.

```
# INSERT INTO mkek (mkek_id, tenant_id, key_algorithm_id, key_status_id, key_
location_id, key_value, display_name)
  VALUES ('<MKEK_ID>','<TENANT_ID>', 1, 2, 1, DECODE('<MKEK_BASE64_ENV_
VARIABLE_NAME>', 'base64'), '<MKEK_NAME>');
```

5. Create the key using the Admin module API. The most recent key created becomes the active key for the corresponding tenant and module. For more information, see Creating keys.

```
# INSERT INTO key (key_id, tenant_id, module_id, display_name, key_status_id,
key_algorithm_id, key_usage)
  VALUES ('<KEY_ID>', '<TENANT_ID>', 1, '<KEK_NAME>', 2, 1, 1);
```

6.3 Configuring symmetric encryption KEKs in KMS mode

6.3.1 Requirements

To use the Stormshield KMaaS with a key management system (KMS), you must meet the following requirements:

- The version of the protocol used for connecting to the KMS must be KMIP 1.4,
- The algorithm used for wrapping KEKs must be AES-GCM.

6.3.2 Generating symmetric encryption KEKs in the KMS

In the interface of the KMS, create a new key with the following values:

name	<name_of_your_kek>
algorithm	AES-256
exportable	true
usage	not necessary

custom attribute	x-sds-kacls-kek-label:<my_kacls_keks_label> Label that identifies all your KEKs. It must match the value of the *kacls_kek_label* field in the *config.json* file.
	x-sds-kacls-tenant-id:<UUIDv4> Identifier of your tenant in UUID v4 format. This ID must match the one specified for the External key service.

The KMIP client must be allowed to use this key.

When the Stormshield KMaaS is being initialized, all KEKs that match the *x-sds-kacls-kek-label* label will be retrieved, regardless of their status in the KMS.

While all retrieved KEKs can be used for unwrap operations, only the most recent key of each tenant will be used for encryption operations. This particular KEK is identified by the *is_active_kek:true* field in logs.

6.3.3 Renewing KEKs in the KMS

For greater security, you can regularly renew the active KEK. To do so, generate a new KEK in the KMS. The Stormshield KMaaS will automatically import this KEK as the active key when the keys are refreshed. Older keys will be kept for decryption operations.

The Thales KMS does not allow more than 200 KEKs to be managed. Please do not exceed this limit.

If the KEK list refresh operation fails, the list of current keys will be kept and service will not be disrupted. The Stormshield KMaaS will refresh the key list again when a periodic or one-off refresh operation is triggered.

7. Securing the Stormshield KMaaS with HTTPS

To secure your Stormshield KMaaS, Stormshield recommends that you:

- Secure connections between the Stormshield KMaaS components with HTTPS.
  In the *config.json* file, fill-in the *https.credentials* section.
  For more information, refer to section https parameter.
- Configure the appropriate TLS ciphers.
  For more information, refer to section Configuring TLS ciphers below.

7.1 Configuring TLS ciphers

When the Stormshield KMaaS is configured in HTTPS, it uses NodeJS which depends on OpenSSL for cryptographic operations.

In RPM mode, by default, the service starts with the following cipher list:

TLS 1.3

- TLS_AES_256_GCM_SHA384
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_SHA256
- TLS_CHACHA20_POLY1305_SHA256

TLS 1.2

- ECDHE-ECDSA-AES256-GCM-SHA384
- ECDHE-RSA-AES128-GCM-SHA256
- ECDHE-RSA-AES256-GCM-SHA384

In Docker mode, the default cipher list is the same as NodeJS 24. This version is based on OpenSSL version 3.5, with a higher security level than in RPM mode (i.e., NodeJS 20 and OpenSSL 3.2).

To communicate more securely when using the Stormshield KMaaS, you can restrict the list of ciphers allowed during TLS operations.

The ciphers used are important security elements. Refer to ANSSI documentation SDE-NT-35 on TLS ciphers.

Note that if you use SELinux to secure the machines, the list of TLS algorithms allowed on the machine may change. This may prevent the Stormshield Key Management as a service from starting or cause incompatibility with external resource retrieval. In this case, you must adjust the list of algorithms allowed by the Stormshield Key Management as a service by following the procedure below.

7.1.1 Modifying the list of TLS ciphers in Docker mode

- Declare the NODE_OPTIONS environment variable:
  `NODE_OPTIONS=--tls-cipher-list=<list-of-tls-algorithms>`

> **EXAMPLE**
> `docker run -e "NODE_OPTIONS=--tls-cipher-list=TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384" stormshield/kmaas:<version>`

7.1.2 Modifying the TLS cipher list in RPM mode

1. Add the *cipher_list.conf* file in the */etc/systemd/system/cse.service.d* directory.
2. Add the following lines in this file:
   `[Service]`
   `Environment=NODE_OPTIONS=--tls-cipher-list=#CUSTOM_CIPHER_LIST#`
   - where -
   `#CUSTOM_CIPHER_LIST#` represents the list of the desired ciphers, separated by ":".

   EXAMPLE
   [Service]
   Environment=NODE_OPTIONS=--tls-cipher-list=TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:

   The cipher format must match the following rules:

   - The Stormshield KMaaS does not support the following cryptographic suites, nor the "!", "+", and "-" operators.
   - You must use the OpenSSL cipher format, and not the standard format.
     To list ciphers with both their standard names and OpenSSL names, run the command:
     `openssl ciphers -stdname`.
     To convert a standard cipher name to the OpenSSL format, run the command:
     `openssl ciphers -convert STANDARD_CIPHER_NAME_TO_CONVERT`.
3. If the list contains valid but not recommended ciphers, a warning log is issued. If a cipher is unknown, an error is issued and the service does not start.

8. Configuring proxy access

If the Stormshield KMaaS is located behind a proxy in your infrastructure, the service must be configured to enable the use of this proxy. To do so, add the URL of the proxy and any exclusions to the configuration file.

8.1 In Docker mode

- Declare the following environment variables to configure the proxy:
  - *https_proxy*: defines the proxy URL,
  - *no_proxy*: defines the endpoints excluded from the proxy.

EXAMPLE
Example of a Docker command declaring environment variables:

```
docker run -v /my-kmaas-config-folder:/etc/stormshield/cse -p 443:3000 -
e https_proxy="https://my-proxy.my-domain" -e
no_proxy="domain.com,192.168.1.10,2001:67c:2e8:22::c100:68b/128"
stormshield/kmaas:<version>
```

8.2 In RPM mode

1. Run the following command:
   ```
   # systemctl edit cse.service
   ```
   The *override.conf* configuration file is created in the */etc/systemd/system/cse.service.d* directory if it was installed in the default directory.
2. Edit the file and copy the following text containing the environment variable for the proxy's URL:
   ```
   [Service]
   Environment="https_proxy=https://my-proxy.my-domain"
   ```

   Where *https://my-proxy.my-domain* is the URL of the proxy used.

3. If you need to exclude certain endpoints from the proxy, declare them in the same file via the no_proxy environment variable. The possible values for this variable are the following:
   - The * character means that all endpoints are excluded. This is equivalent to disabling the proxy.
   - A domain, for example *domain.com*,
   - A domain suffix, for example *.domain.com*,
   - A v4 or v6 IP address, for example *192.168.1.10* or *2001:67c:2e8:22::c100:68b*,
   - A v4 or v6 IP address in CIDR, for example *172.30.0.0/16* or *2001:67c:2e8:22::c100:68b/128*.

   The different values must be separated by commas.

   EXAMPLE
   Example of a file `cse.service` in which the proxy is configured and different endpoints are excluded from the proxy:
   ```
   [Service]
   Environment="https_proxy=https://my-proxy.my-domain"
   Environment="no_
   proxy=domain.com,192.168.1.10,2001:67c:2e8:22::c100:68b/128"
   ```

4. Reload the systemd service using the following command:
   ```
   # systemctl daemon-reload
   ```

5. Start the systemd service using the following command:
   ```
   # systemctl start cse
   ```
   A startup log indicates that the service is launched in proxy mode. For more information, refer to *Stormshield KMaaS Log Guide*.

9. Backing up and restoring the Stormshield KMaaS files

In Docker mode, you must configure your orchestrator to guarantee data persistence. Implement a regular file backup strategy, as well as secure file storage on a separate infrastructure.

In RPM mode, Stormshield recommends that you deploy Stormshield KMaaS in a cluster to improve performance and limit the impact if a failure occurs. However, some of the files used by the service and the configuration of the server must be backed up if you have deployed the solution via an RPM.

9.1 Backing up Stormshield KMaaS files

- Back up the files below **every time changes are made**:

Name	Description	Level	Consequences if file is lost
config.json	File that describes the configuration of the Stormshield KMaaS server	Moderate	Configuration must be rebuilt
keks.json	**File containing all the KEKs**	**Very high**	**Users' encrypted data cannot be decrypted**
HTTPS certificates	Files used for the HTTPS connection	Moderate	Configuration must be rebuilt
KMS certificates	Files used for the KMS connection	Moderate	Configuration must be rebuilt

9.2 Restoring the files in Docker mode

In Docker mode, the various configuration files are made available to containers via a volume. If one of the containers fails, recreate the Stormshield KMaaS container so that the configuration files are automatically taken into account.

For more information, please refer to your orchestrator documentation.

9.3 Restoring the files in RPM mode

If any node of the cluster fails, follow the procedure below to restore files on each node:

1. Disconnect the node from the cluster.
2. Reconfigure the Red Hat instance if necessary.
3. Reinstall the service if necessary.
4. Deploy the backed up files:
   - config.json
   - keks.json
   - Certificates for HTTPS
   - Certificates for KMS

5. Restart the service and check whether it runs.
6. Reconnect the node(s) of the cluster.

10. Key Access Control List Service (KACLS)

The KACLS is a solution in which corporate data managed in the Google Workspace ecosystem can be protected, edited and consulted. Google Workspace is Google's cloud-based application suite for professionals. For more information, refer to the Google Workspace documentation.

The KACLS relies on Google Client Side Encryption (CSE), the end-to-end encryption method that Google offers for its Google Workspace applications. CSE is configured in the Google administration console. This technology is available only on Chrome or Microsoft Edge (Chromium) browsers. For more information on supported browsers, refer to the Google Client Side Encryption documentation, section *Browser requirements*.

Google generates DEKs (Data Encryption Keys) to encrypt files. Before such keys are stored on Google servers, the Stormshield KMaaS wraps them using KEKs (Key Encryption Keys).

Stormshield KMaaS is installed in your on-premise or cloud-based infrastructure; KEKs are therefore stored with you and never sent to Google servers.

Before performing cryptographic operations, the KACLS first conducts a double verification:

- Authentication: checks the identity of the user requesting the operation,
- Authorization: checks the user's access privileges for the file to encrypt/decrypt.

The KACLS generates logs for all the operations that it performs.

NOTE
The use of the solution in any way other than as described in the documentation is not managed. Alternatively, get in touch with Stormshield Support for clarification.

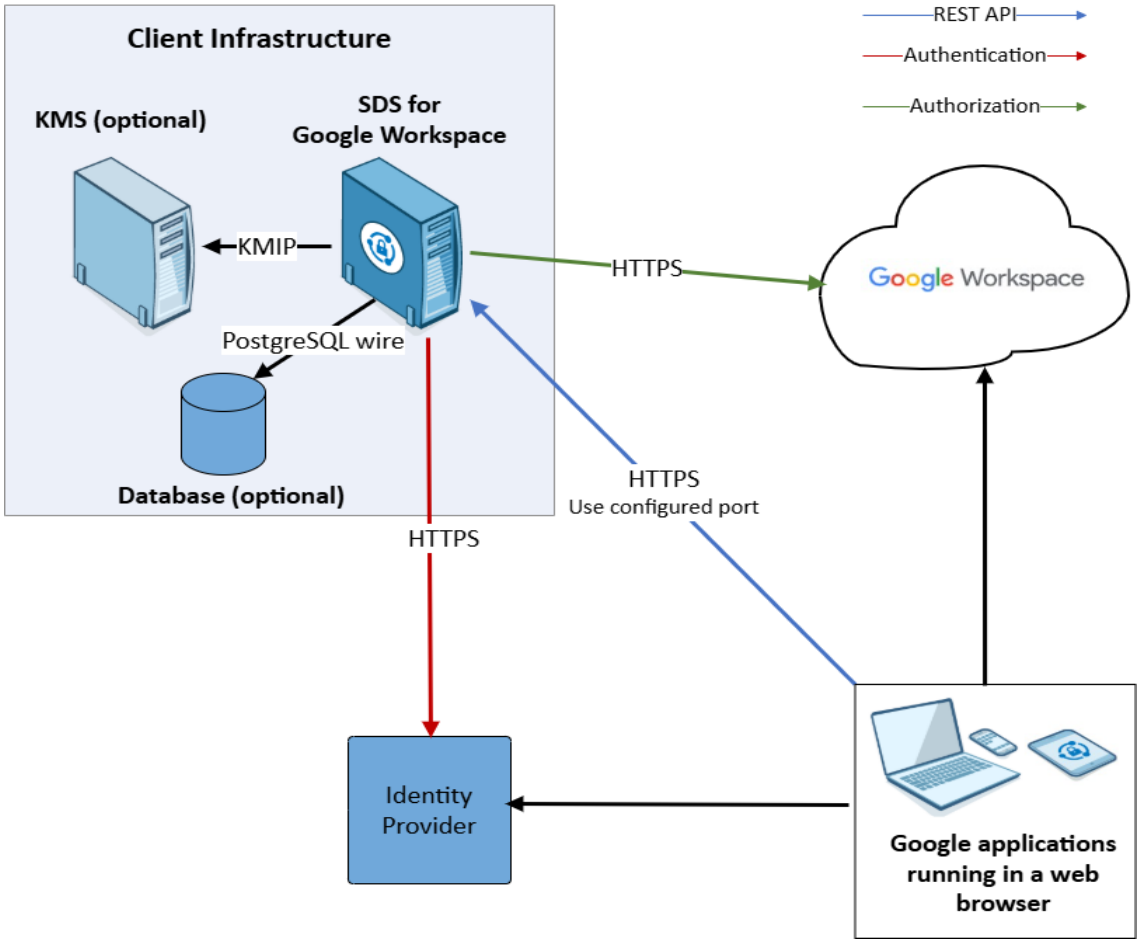
10.1 Understanding the requirements

10.1.1 Global requirements and recommendations on administrators and network rules

For information, refer to section Understanding the global requirements

10.1.2 Network requirements

The diagram and the table below describe the various streams of incoming and outgoing traffic on the KACLS server. Configure your network to allow the following connections:



Description	Protocol	Source	Source port	Destination	Destination port
The KACLS REST API	HTTPS	Google application	*	Stormshield KMaaS	Depends on the administrator's configuration (*config.json*)
Getting the configuration of OpenID authentication	HTTPS	Stormshield KMaaS	*	OpenID endpoint	Specified by the configuration of the authentication service (usually 443)
Getting the configuration of the JWKS authorization	HTTPS	Stormshield KMaaS	*	JWKS endpoint	Specified by the configuration of the authorization service (usually 443)
Getting decryption keys	KMIP version 1.4	SDS for Google Workspace	*	Client's infrastructure	Depends on the administrator's configuration (usually 5696)
Getting keys when *persistence_type* is set to *database*	PostgreSQL wire protocol	Stormshield KMaaS	*	Client's infrastructure dedicated to Postgres database	Depends on the administrator's configuration (usually 5432)

NOTE
In Docker deployment mode, you must expose the ports to the containers. For more information, please refer to your orchestrator documentation.

The content of the requests processed by the KACLS is in JSON format only. You can add the following rules to your web firewall (WAF) or load balancer to ensure optimum protection:

- All HTTP requests are blocked except:
  - The POST requests with a *Content-Type* header including "application/json".
  - The GET and OPTIONS requests without a *Content-Type* header or with a *Content-Type header* including "application/json".

10.2 Deploying the KACLS infrastructure

The table below lists the various steps involved in deploying the KACLS.

Click on a link to open the corresponding procedure in this guide.

Steps	Description
1	Configuring the identity provider and Configuring Google Workspace Client-side encryption
2	Setting up the infrastructure and install the Stormshield KMaaS
3	Configuring the config.json file and Configuring the KACLS
4	Configuring Key Encryption Keys (KEKs) • Configuring keys in KMS mode - or - • Configuring the keks.json file
5	Setting up a network configuration that can be reached via Google Workspace
6	Running the Stormshield KMaaS
7	Checking system health
8	Configuring load balancing
9	(Optional) Using the Stormshield KMaaS for Google Workspace in HTTPS and Configuring TLS ciphers
10	Setting up logging
11	(Optional) Customizing the authorization rules by applying an OPA policy and Customizing the authorization rules

10.2.1 Setting up the Stormshield KMaaS in Google Workspace

In the table below are the various steps involved in adding Stormshield KMaaS in Google Workspace. Click on a link to open the corresponding procedure in the Stormshield KMaaS guide or in Google help.

Steps	Description
1	Setting up your external key service
2	Connecting Google Workspace to the external key service
3	Connecting Google Workspace to the Identity provider
4	Indicating the well-known file in the config.json file
5	Enabling the service for users: Drive, Meet, Calendar, Gmail

10.3 Configuring the KACLS

The KACLS is configured in the *kacls* section of the *config.json* file. You can configure it independantly for each tenant. The template for KACLS configuration block is as follows:

```
"kacls": {
       "enable": true,
       "user_authentication": {
         "enable_wellknown_cse_discovery": "_ENABLE_WELLKNOWN_CSE_DISCOVERY_",
         "idps": [
           {
             "discovery_uri": "_AUTHENTICATION_OPEN_ID_CONFIGURATION_URL_",
             "client_id": "_AUTHENTICATION_AUDIENCE_"
           },
           {
             "jwks_uri": "_IDPS_JWKS_URL_",
             "audience": "_IDPS_AUDIENCE_",
             "issuer": "_IDPS_ISSUER_"
           }
         ]
       },
       "admin_authentication": [
         {
           "discovery_uri": "_ADMIN_AUTHENTICATION_DISCOVERY_URI_",
           "client_id": "_ADMIN_AUTHENTICATION_ISSUER_"
         }
       ],
       "wrapprivatekey_authentication": [
         {
           "discovery_uri": "_WRAPPRIVATEKEY_AUTHENTICATION_DISCOVERY_URI_",
           "client_id": "_WRAPPRIVATEKEY_AUTHENTICATION_ISSUER_"
         }
       ],
       "migration": {
         "enable": "_ENABLE_",
         "kaclstokacls_token": {
           "kid": "_KACLS_TO_KACLS_KEY_",
           "format": "_FORMAT_",
           "key": "_KEY_",
           "duration": "_DURATION_"
         },
         "acls": {
           "kacls_urls": ["_ALLOWED_KACLS_URL_"]
         }
```

```
    },
    "delegate": {
      "enable": "_ENABLE_",
      "authentication": {
        "key": "_PRIVATE_KEY_BASE64_"
      }
    },
    "keys": {
      "users_private_keys": {
        "crypto_backend": {
          "id": "_CRYPTO_BACKEND_ID_"
        }
      }
    },
    "crypto_backends": [
      {
        "id": "_CRYPTO_BACKEND_ID_",
        "name": "_CRYPTO_BACKEND_NAME_",
        "type": "_CRYPTO_BACKEND_TYPE_",
        "configuration": {
          "host": "_HOST_",
          "port": "_PORT_",
          "vendor": "_VENDOR_",
          "model": "_MODEL_",
          "credentials": {
            "key": "_KEY_",
            "cert": "_CERT_",
            "ca": "_CA_"
          },
          "domain_id": "_KMS_CRYPTO_BACKEND_DOMAIN_ID_"
        }
      }
    ],
    "policy_enforcement": {
      "enable": false,
      "type": "_POLICY_ENFORCEMENT_TYPE_",
      "opa_server": {
        "url": "_URL_",
        "authentication": {
          "type": "basic",
          "user_id": "_USER_ID_",
          "password": "_PASSWORD_"
        }
      }
    }
  },
```

Parameter	Description	Type	Optional/ mandatory
enable	Enables or disables the KACLS module, KACLS.	Boolean	Mandatory to use the KACLS module.

user_authentication parameters

Object containing the configurations that allow the client to authenticate.

Parameter	Description	Type	Optional/ mandatory
enable_wellknown_cse_ discovery	Activated by default (true). Enables the use of the .well-known remote configuration to validate user authentication.	Boolean	Optional

Parameter	Description	Type	Optional/ mandatory
idps	Object array containing the configuration of the identity providers for client authentication. It must include either both elements "discovery_uri" and "client_id", or the three elements "jwks_uri", "audience" and "issuer": • discovery_uri: URL to the OpenID JSON configuration file, • client_id: recipient of the JWT authentication token (see RFC 7519), • jwks_uri : URL to the JSON Web Key Set file, • audience: recipient of the JWT authentication token (see RFC 7519), • issuer: issuer of the JWT authentication token (see RFC 7519). An entry must be added for each identity provider. For more information on the values of the elements, see Retrieving import values.	Array	Optional

The values of the authentication parameters depends on the method used:

- OpenID configuration file (for OneLogin for instance):
  - `_AUTHENTICATION_OPEN_ID_CONFIGURATION_URL_` is the URL for the OpenID configuration file.
    For OneLogin authentication, it must resemble:
    https://<domain>.onelogin.com/oidc/2/.well-known/openid-configuration.
    For Google authentication, it is similar to:
    https://accounts.google.com/.well-known/openid-configuration.
  - `_AUTHENTICATION_AUDIENCE_`
    For OneLogin authentication, it corresponds to the *audience* setting.
    For Google authentication, it corresponds to the *OAuth Client ID* setting.
- JSON Web Key Set file (JWKS):
  - _IDPS_JWKS_URL_ corresponds to the URL for the JWKS file that contains the signature and/or encryption keys.
    For Google authentication, it is similar to:
    https://www.googleapis.com/service_accounts/v1/jwk/gsuitecse-tokenissuer-drive@system.gserviceaccount.com
  - _IDPS_AUDIENCE_
    For Google authentication, it corresponds to the *OAuth Client ID* setting.
  - _IDPS_ISSUER_ corresponds to the issuer of the authentication token.
    For Google authentication, it is similar to gsuitecse-tokenissuer-drive@system.gserviceaccount.com

For more information, see section Retrieving import values

admin_authentication parameter

JSON object array describing how to validate the authentication of administration routes via a local configuration.

Parameter	Description	Type	Optional/ mandatory
discovery_uri	URL to the OpenID JSON configuration file.	String	Optional
client_id	Recipient of the JWT authentication token (see RFC 7519). An entry must be added for each identity provider.	String	Optional

wrapprivatekey_authentication parameter

JSON object array describing how to validate the authentication of */wrapprivatekey* routes via a local configuration. For more information, see Migrating an external key service to another.

Parameter	Description	Type	Optional/ mandatory
discovery_uri	URL to the OpenID JSON configuration file.	String	Optional
client_id	Recipient of the JWT authentication token (see RFC 7519). An entry must be added for each identity provider.	String	Optional

migration parameter

JSON object array containing information for the migration of a KACLS to another or the use of a backup KACLS.

Parameter	Description	Type	Optional/ mandatory
enable	Enables or disables the migration from one KACLS to another.	Boolean	Optional
kaclstokacls_token	• kid: identifier used to generate a JWKS. • format: format of the key (PEM). • key: private key in PEM format. Used to form JWT authentication tokens and generate a JWKS making it possible to check these tokens. • duration: lifetime of the generated JWT authentication token.	Array	Mandatory if enable is set to true
acls	• kacls_urls: list of allowed KACLS URLs. Must begin with "*https://*".	Array	Mandatory if enable is set to true

crypto_backends parameter

JSON object array containing the definition of the backend component performing the cryptographic operations.

When using a crypto backend of type "kms" with a KMS domain, the *domain_id* values of all the tenant configurations must imperatively be the same and match the domain used by the KMIP configuration.

For more information, refer to section kmip_configuration parameters and the Thales documentation.

WARNING
Stormshield does not guarantee the proper functioning of the product if you do not follow these instructions.

Parameter	Description	Type	Optional/ mandatory
id	ID of the cryptographic backend in the form of a UUID v4 that you generate.	String in UUID format	Mandatory
name	Name of your choice for the cryptographic backend.	String	Mandatory
type	Cryptographic backend type. The possible values are: • kms to use the KMS API • node to use the KACLS	String	Mandatory
configuration: JSON object array containing the cryptographic backend configuration in the "kms" mode.			
host	URL of the KMS	String	Mandatory in "kms" mode"
port	KMS port	String	Mandatory in "kms" mode"
vendor	KMS vendor (Thales)	String	Mandatory in "kms" mode"
model	KMS model (Ciphertrust)	String	Mandatory in "kms" mode"
credentials	It includes the following fields: • ca_certificate_path: path to the certification authority. • client_certificate_path: path to the KMS client certificate. • client_private_key_path: path to the KMS user key.	String	Mandatory in "kms" mode"

Parameter	Description	Type	Optional/ mandatory
domain_id	UUID v4 pointing to the Ciphertrust KMS domain to be used by the application. Only used for the KMS REST API client. For the KMIP client, you must define the KMS domain in the mTLS certificate (See kmip_ configuration parameters). If not provided, the root domain is used. **WARNING** If you want to use the root domain, do not add its UUID in the configuration file as it is not UUID v4 compliant and will generate errors. Simply remove the domain_id setting from the file.	String	Optional

Configuration of the cryptographic backend

A cryptographic backend is a tool used by the CSE for Gmail, allowing to choose the type of encryption and signature that the KACLS will apply, including the Gmail private keys. Two backend modes are available: 'node' for the KACLS, and 'kms' for the KMS.

The supported algorithms for each mode are:

	Decryption algorithm	Signature algorithm
'node' mode	RSA/ECB/OAEPwithSHA-1andMGF1Padding, RSA/ECB/OAEPwithSHA-256andMGF1Padding, RSA/ECB/OAEPwithSHA-512andMGF1Padding	SHA1withRSA/PSS, SHA256withRSA/PSS, SHA512withRSA/PSS, SHA1withRSA SHA256withRSA SHA512withRSA
'Node' mode with the CVE-2023-46809 enabled (See the limitations below)	RSA/ECB/PKCS1Padding, RSA/ECB/OAEPwithSHA-1andMGF1Padding, RSA/ECB/OAEPwithSHA-256andMGF1Padding, RSA/ECB/OAEPwithSHA-512andMGF1Padding	SHA1withRSA/PSS, SHA256withRSA/PSS, SHA512withRSA/PSS, SHA1withRSA SHA256withRSA SHA512withRSA
'kms' mode	RSA/ECB/PKCS1Padding	SHA1withRSA/PSS, SHA256withRSA/PSS, SHA512withRSA/PSS, SHA1withRSA, SHA256withRSA SHA512withRSA

- With the 'kms' mode, be aware of the following:
  - The KACLS is only compatible with the API of the Ciphertrust Manager from Thales.
  - Only one version of the private keys associated with a Ciphertrust keyname is supported. You should therefore not use the versioning feature of the Thales Ciphertrust Manager KMS for the encryption or signature keys used for the KACLS for Gmail.
  - For maximum security and flexibility, you can use a specific KMS key domain (Ciphertrust domains) via the domain_id parameter. In this case, also verify that dependent SDS products share the same domain (e.g., SDS Orchestrator). For more information on this configuration, refer to the Thales documentation and contact your KMS administrator.
  - Since PKCS 1.5 message signing is not compatible with Ciphertrust's REST API, the KACLS uses the KMIP protocol to perform signing operations. Configure the section kmip_configuration of the *config.json* file to sign messages in PKCS 1.5.
- In 'node' mode, data encryption using the PKCS 1.5 algorithm is vulnerable, particularly to the *Marvin Attack*. NodeJS version 20.11.1 has therefore removed the use of this algorithm via CVE-2023-46809. As Google only supports PKCS 1.5 for message signature and encryption, you must disable this CVE in NodeJS for the KACLS to use this feature.
  To do so, **in RPM mode**:
  1. Open the */etc/systemd/system/cse.service* file.
  2. Replace the `ExecStart=/usr/bin/env node cse`
     - by -
     `ExecStart=/usr/bin/env node --security-revert=CVE-2023-46809 cse`

     This bypass is not useful if you are in 'kms' mode.

In Docker mode, the product is not vulnerable as it uses NodeJS version 24.

NOTE
If the HTTP proxy is enabled, you must exclude the KMS domain from the proxy via the *no_proxy* environment variable. For more information, see the section Configuring proxy access.

Below is an example of a cryptographic backend configuration in "kms" mode with the Thales Ciphertrust Manager KMS:

```
"crypto_backends": [
  {
    "id": "3711cab6-83fc-4a97-9438-a1500edfd01a",
    "name": "My crypto tool",
    "type": "kms",
    "configuration": {
      "host": "https://web.ciphertrustmanager.local",
      "model": "ciphertrust",
      "vendor": "thales",
      "port": 443,
      "credentials": {
         "ca": "/etc/stormshield/cse/ca_kms.pem",
         "cert": "/etc/stormshield/cse/cert_kms.pem",
         "key": "/etc/stormshield/cse/key_kms.pem"
      }
    }
  }
],
"keys": {
    "users_private_keys": {
      "crypto_backend": {
```

```
            "id": "3711cab6-83fc-4a97-9438-a1500edfd01a"
        }
    }
}
```

keys parameter

Object containing the UUID of the cryptographic backend to be used for cryptographic operations.

Parameter	Description	Type	Optional/ mandatory
users_private_keys	• crypto_backend: object defining the cryptographic backend to be used to get private keys. - id: UUID of the cryptographic backend defined in the "crypto_backend.id" object.	String	Mandatory if the crypto_ backend object is configured

delegate parameter

JSON object array containing the definition of the *delegate* software component performing the delegation operations.

Parameter			
authentication	JSON object allowing to sign JWT tokens in RS256. It includes the following field: • key: private key signed in base64 and used to sign authentication token.	Object	Mandatory if the enable field is set to true
enable	Enable the delegation feature.	Boolean	Mandatory

policy_enforcement parameter

JSON object containing the configuration of the OPA Policy for the KACLS. For more information about OPA Policy, refer to Implementing the authorization rules with Open Policy Agent.

Parameter			
enable	Enable the use of OPA rules for the module.	Boolean	Mandatory for each module enabled except the PKI.
type	Kind of OPA policy to use. The possible values are: • opa_local: this mode uses local files *policy.wasm* and *policy.data.json*. The file names must be adapted according to the module used. • opa_server: this mode uses a remote OPA server.	String	Mandatory if policy_ enforcement.enable is set to true
opa_server: JSON object describing the parameters required to access the OPA policy server. Stormshield guarantees compatibility with OPA version 1.2.0.			

Parameter			
url	URL of data API exposed endpoints. For more information, see OPA documentation. *Example*: If your rego package is *stormshield.kmaas* and you have the *allow* variable in this package, your url will be: *https://opa-server/v1/data/stormshield/kmaas/allow* The authorized protocols are http and https. Stormshield strongly recommends https in production.	String	Mandatory if policy_enforcement.type is set to opa_server
authentication	JSON object describing the parameters required to authenticate to the OPA policy server. It includes the following fields: • type: Type of authentication used to connect to the policy server. The prescribed value is "basic". • user_id: Identifier of the user account used to connect to the policy server. Mandatory if authentication.type is set on "basic" • password: Password of the user account used to connect to the policy server. Mandatory if authentication.type is set on "basic"	Object	Mandatory if type is set to opa_server

For more information about Open Policy Agent, refer to section Implementing the authorization rules with Open Policy Agent and Customizing the authorization rules.

10.4 Checking system health

After you have installed and run the Stormshield KMaaS, check that it is running correctly.

1. Use the *status* API route:

```
curl -H "Origin: <origin_url>" <my-cse-full-url>/status
```

where:

Parameter	Value
<origin_url>	The Stormshield KMaaS enforces across-origin resource sharing (CORS) rule to guarantee that requests genuinely originate from the Google API. This rule makes it possible to verify the *origin* HTTP header. The origins that are allowed are all URLs of https://*.google.com type, for example: • https://client-side-encryption.google.com • https://admin.google.com • Requests containing an incorrect *origin* header are rejected. For more information, refer to the Google documentation Connect to your identity provider for client-side encryption.
<my-cse-full-url>	Full URL specified for the external key service. For more information, refer to Specifying the External key service.

EXAMPLE

```
curl -H "Origin: https://client-side-encryption.google.com"
https://cse.example.com/api/v1/a4670b0-4bc11-4290-a5bd-
498c2e1fb0b/status
```

2. If the system is running correctly, the *status* API return must be in the following form:

```
{"server_type":"KACLS","vendor_
id":"Stormshield","version":"4.3.0.2354","name":"name of my
CSE","operations_supported":
["wrap","unwrap","digest","rewrap","privilegedwrap","privilegedunw
rap","wrapprivatekey","privatekeysign","privatekeydecrypt","privil
egedprivatekeydecrypt"]}
```

10.5 Configuring the identity provider

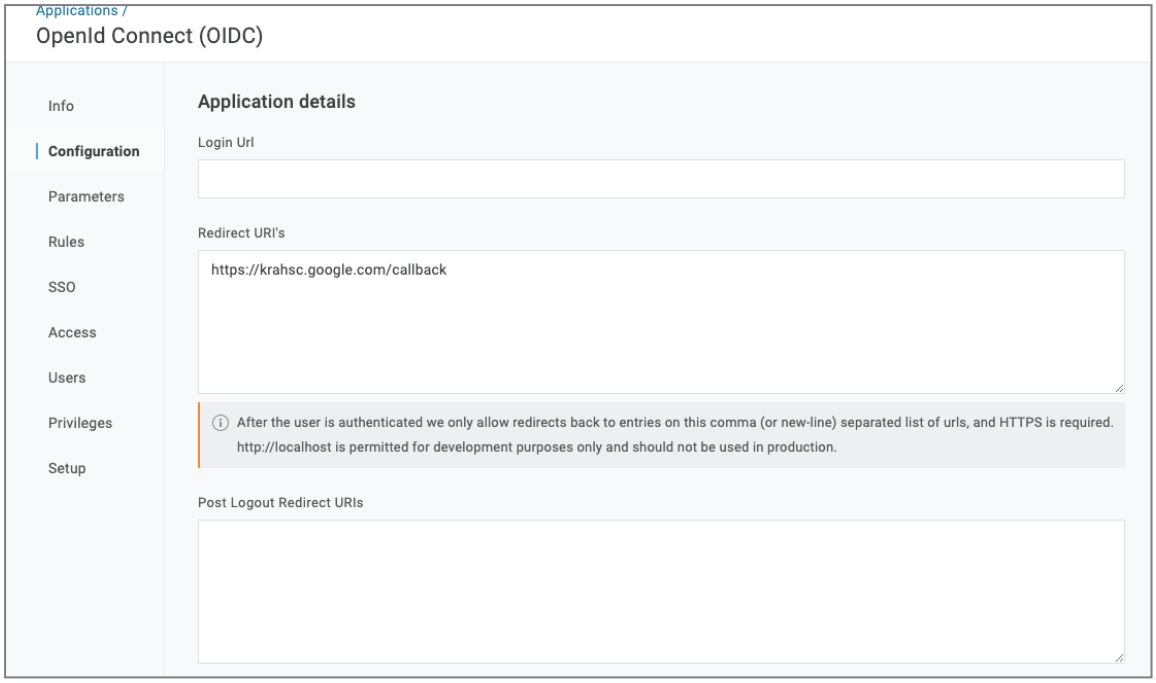
The Stormshield KMaaS uses an identity provider (IDP) to authenticate end users, manage their access permissions and their life cycles. Configure the provider of your choice and create an OpenID Connect application.

The Stormshield KMaaS is compatible with JWT tokens signed with the RS256 algorithm.

The procedure below describes the configuration with One Login. For more information, refer to the One Login documentation.

10.5.1 Specifying the redirect URL

In OpenID Connect, select the **Configuration** menu, then specify the redirect URI in the **Redirect URI's** field. For more information, refer to the Google documentation Connect to your identity provider for client-side encryption.



10.5.2 Retrieving import values

In OpenID Connect, select the **SSO** menu and take note of the **ClientID** and **Issuer URL** import values:

These values will be used in the *config.json* file, in the **tenants** > **user_authentication** > **idps** section of the Stormshield KMaaS. Below are a few examples:

- **ClientID**: `3e14f1a0-5814-0550-cy6e-0bd6abe5ty43540000`
- **Issuer URL** (*Well-known* configuration): https://stormshield-example.onelogin.com/oidc/2/.well-known/openid-configuration

For more information on declaring the identity providers in the Stormshield KMaaS, refer to section Configuring the Stormshield KMaaS, tenants parameter.

10.5.3 Managing authentication tokens

According to the specifications provided by Google, the authentication token contains a JSON Web Token (i.e., JWT). For more information, see the RFC7516 document.

The mandatory and optional fields expected by the KACL depending on the routes used are listed in the following table:

Routes	Mandatory fields	Optional fields
• wrap • unwrap • privilegedwrap, • privilegedunwrap • privatekeydecrypt, • privatekeysign, • privilegedprivatekeydecrypt • privilegedprivatekeysign	• iss • aud • exp • iat • email	• google_email
Authentication token to the KACLS: • privilegedunwrap They are used to authenticate a KACLS to another one in the context of a migration.	• iss • aud • exp • iat • kacls_url • resource_name	
delegate	• iss • aud • exp • iat • email • delegated_to • resource_name	
wrapprivatekey	• iss • aud • exp • iat	

For more information, refer to the Google documentation.

Authentication tokens related to delegation

The authentication tokens used by the encryption and decryption operations (i.e., wrap and unwrap routes) in the context of a delegation operation are dynamically generated by the Stormshield KMaaS (delegate route): for security reasons, and as recommended by Google, these tokens have a lifetime of 15 minutes.

10.6 Configuring Google Workspace Client-side encryption

You must indicate the URL of the external key service and the identity provider in the Google Workspace administration console.

For more information, refer to the Google documentation Use client-side encryption for users' data.

10.6.1 Specifying the External key service

External key service is the section in the Google Workspace administration console in which you specify information for the KACLS.

With the KACLS, several external key services can be used in your Google Workspace tenant's administration console. For example, if you want separate services for each distinct organizational unit (OU) in your organization for various Google applications (Meet, Drive, ...).

In standalone mode, you must enter a UUID for every tenant installed so that their associated KEKs will be available. For further information, refer to the section Adding KEKs to the file.

If you are using a Key Management System (KMS), the tenant's UUID is included in the attributes of the KEK. For more information, refer to the section Configuring symmetric encryption KEKs in KMS mode.

With the KACLS, several tenants can be used on the same instance of the encryption service. For example, if your organization has several domains, you can manage each tenant independently for each domain.

An external key service must be specified for each tenant.

- The **Name** of the external key service can be shown in error messages that the end user will see.
- The **URL** of the external key service consists of the following:

Address of the Stormshield KMaaS instance that you are installing	E.g., https://cse.example.com/api/v1
Tenant UUID	E.g., a4670b0-4bc11-4290-a5bd-498c2e1fb0bf You must generate a v4 UUID to identify tenants, even when there is only one on your instance.

EXAMPLE
https://cse.example.com/api/v1/a4670b0-4bc11-4290-a5bd-498c2e1fb0b

Google applications will use this URL, so it must be a public address.

10.6.2 Specifying the identity provider (IDP)

For more information, refer to the Google documentation Connect to your IdP for CSE.

10.7 Using remote authentication

You can create a remote configuration file, *cse-configuration*, to share your authentication credentials with external collaborators. This file must be in a directory */.well-known/*, located at the root of the domain (*https://cse.${domain}/.well-known/*). It makes it possible to verify the signature of the user's token and indicate which identity providers to use.

The remote file will be looked up if the *user_authentication* section in the *config.json* file is not filled in. It is retrieved during authentication via the URL:
https://cse.${domain_from_email_from_token}/.well-known/cse-configuration

This is a fixed URL. Ensure that it can be reached by using the Stormshield KMaaS.

NOTE
For security reasons, the routes *privilegedwrap*, *privilegedunwrap*, *privilegedprivatekeydecrypt*, and *wrapprivatekey* are not allowed for remote authentication.

For more information, refer to the Google documentation Connect to identity provider for client-side encryption website.

To create the remote authentication file:

- Create a file named *cse-configuration*. Its contents are as follows:

```
{
"name": "_IDP_NAME_",
"client_id": "_AUTHENTICATION_AUDIENCE_",
"discovery_uri": "_AUTHENTICATION_OPEN_ID_CONFIGURATION_URL_",
"grant_type": "_GRANT_TYPE_"
}
```

Parameter	Description	Type	Authorized values	Optional/ mandatory
name	Name of the identity provider.	String		Optional
client_id	OIDC (OpenID Connect) client ID that the client application uses to get a JWT.	String		Mandatory
discovery_uri	OIDC discovery URL, as defined in the OpenID specification.	String		Mandatory
grant_type	OAuth traffic used for OIDC	String	implicit | authorization_code	Optional

If you use the Google identity provider, the values of the authentication settings are as follows:

```
{
"name": https://accounts.google.com
"client_id": "37*********",
"discovery_uri": "https://accounts.google.com/.well-known/openid-configuration"
}
```

10.8 Using the KACLS with Drive, Meet and Calendar

The Stormshield KMaaS allows users to encrypt data for the following Google applications:

Application	Encryption perimeter	Availability	Use
Google Drive	Encrypting Google Drive confidential documents.	• Windows and macOS desktops (Google Drive for Desktop) • iOS and Android mobile devices	• *well-known* remote file • Local configuration

Application	Encryption perimeter	Availability	Use
Google Meet	Encryption of video conferences and calls created with Google Meet	• Google Workspace web client • iOS and Android mobile devices	• *well-known* remote file • Local configuration
Google Calendar	Encrypting a meeting created with Google Calendar: related description, attachments, and Meet conference.	• Google Workspace web client • iOS and Android mobile devices	• *well-known* remote file • Local configuration

10.8.1 Importing sensitive external files to Google Drive (Beta)

The KACLS supports the bulk importing of sensitive data into Google Drive. Data imported from third-party storage are encrypted by the KACLS in Google Drive.

For more information, see the Google documentation.

This Beta feature is currently under development at Google.

10.8.2 Using data loss prevention rules for Google Drive (Beta)

Using Google data loss prevention (DLP), you can create and apply rules to control the content that users can share in Google Drive files outside the organization. DLP gives you control over what users can share, and prevents unintended exposure of sensitive information.

From the Google Admin console, you can set up policy-based actions and KACLS automatically encrypts Drive files when sensitive content is detected.

A Google Workspace Assured Controls license is required to use a DLP rule for each user.

To set up a rule that automatically encrypts files, please follow Google's procedure, described in the Google documentation. This Beta feature is currently under development at Google.

Whenever a DLP rule is triggered on an unencrypted file, the */systemwrap* route is directly called by Google and encrypts the file without any user action.

For more information on the */systemwrap* route, refer to section Inputs specific to the digest and systemwrap API routes.

10.8.3 Enabling Google Meet hardware use

To attend an encrypted Google Meet conference from a room equipped with Meet hardware, the user must delegate authentication to the Meet hardware device.

Whenever the users try to connect to an encrypted Google Meet, they are asked to scan the meeting QR code with their phone, using only the Google Lens application. Once authenticated via their phone, they are connected to the encrypted meeting.

To enable delegation:

- In the local configuration file *config.json*, declare your authentication information in the section *tenantid - delegate*.

For more information, refer to section delegate parameter.

10.8.4 Enabling external user access for Google Drive and Google Meet

You can share encrypted content with external users in Google Drive and invite external participants to encrypted Google Meet conferences.

This feature will be available on mobile applications in a future release of the Stormshield KMaaS.

To enable the Guest Access feature:

1. Create a dedicated identity provider for Google Drive and declare all desired external users so that they can authenticate. Only declared users will be able to access shared encrypted content. For more information, see Configuring the identity provider.
2. In the same way, create a dedicated identity provider for Google Meet.
3. In the Google Workspace administration interface, add the identity providers specific to external users. For more information, refer to the Google documentation on configuring a guest IdP for all external users. In the **Configure guest IdP** section, select the **Drive and Doc** and/or the **Meet** options.

Once the Guest Access feature has been enabled:

- When a document is shared via Google Drive to external users, they receive emails enabling them to connect to the dedicated identity provider and view the document.
  If the external users do not have a Google account, they must also validate their email addresses with Google every 7 days.
- When you invite external users to an encrypted Google Meet conference, they receive emails containing a link to join the encrypted conference directly. To do this, they must authenticate to the dedicated identity provider.

If using the Guest Access feature, it is not yet possible to share data between participants using a well-known file.

10.8.5 Enabling the use of a Google application via a remote file

- Configure your identity provider as described in the Google documentation Connect to your identity provider for client-side encryption.

10.8.6 Enabling the use of a Google application in the local configuration

- In the *config.json* local configuration file, declare the *client_id* of your applications in the *user_authentication - idps* section.

For example, if you use the Google Identity provider, this section of the *config.json* file should be as follows for drivefs, drive-android, and drive-ios:

```
"user_authentication": {
  "idps": [
  {
    "discovery_uri": "https://accounts.google.com/.well-known/openid-configuration",
    "client_id": "947318989803-
k88lapdik9bledfml8rr69ic6d3rdv57.apps.googleusercontent.com"
  },
  {
    "discovery_uri": "https://accounts.google.com/.well-known/openid-configuration",
    "client_id": "378076965553-
g44pde5vvf113hdd8j84a32kl4e7hqa0.apps.googleusercontent.com"
```

```
    },
    {
      "discovery_uri": "https://accounts.google.com/.well-known/openid-configuration",
      "client_id": "640853332981-
r48oo8ht2kl9v029vsgtatkh4gtue0pn.apps.googleusercontent.com"
    }
    ]
},
```

For more information, refer to the user authentication parameters section.

Currently, sharing encrypted content with external users is only available on web applications. This feature will be available on mobile applications in a future release.

10.9 Decrypting files and emails

To decrypt files and emails encrypted by the Stormshield KMaaS, you can use Google's decryption utility *decrypter.exe* (Beta version):

1. As the super-administrator of the Google Workspace domain, export encrypted data using the export tool or Google Vault. For more information, refer to the Google documentation. The export operation is only possible once every 30 days.
2. Decrypt encrypted data using Google's *decrypter.exe* utility (Beta version). For more information, refer to the Google documentation.

10.10 Using the KACLS with Gmail

The KACLS can be used with Gmail. This feature is available with the web version of Gmail and with the mobile application on Android and iOS.

Two modes are available:

- Gmail standard mode with encrypted keys stored at Google,
- Gmail advanced mode based on a Key management system (KMS) with keys stored in the KMS.

Depending on the mode you choose, you need to know the limitations about the supported algorithms. For more information, refer to the table in the section Configuration of the cryptographic backend.

Stormshield recommends the use of different key pairs for encryption and signature. In this case, repeat the step Encrypting users' private keys with the KACLS for each private key.

To help you using of Gmail with the client-side encryption service, you can implement the SDS Orchestrator solution. It provides and manages encryption and signature keys for your Google Workspace accounts. For more information, please contact your Stormshield sales representative.

10.10.1 Using Gmail in standard mode

Configuring the KACLS

Modify the *config.json* file as described in the steps below: For more information, refer to the Configuring the Key Access Service section.

1. Fill in the `crypto_backend` section, and assign the "node" value to the `type` field. Do not fill in the `configuration` block. See crypto_backends parameter and Configuration of the cryptographic backend.
2. In the `id` field of the `keys` section, enter the UUID of the cryptographic backend set in step 1. See keys parameters and Configuration of the cryptographic backend.
3. Fill in the `authorization` section with Gmail information as shown in Authorization settings.
4. Fill in the `wrapprivatekey_authentication` section as shown in wrapprivatekey_authentication parameters.
5. Optional. Fill in the `admin_authentication` section as shown in admin_authentication parameters to perform privileged operations.

Encrypting users' private keys with the KACLS

Ensure that the KACLS is fully configured and operational before following the steps below.

For every private key to be encrypted, call up the */wrapprivatekey* API route in POST with the following headers and payload:

- **URL**: in the format "{protocol: http | https}://{kacls url}/api/v1/{tenantId}/wrapprivatekey", where:
  - {kacls url} is the URL of the key service that you have declared in Specifying the External key service
  - {tenantId} is your tenant's UUID.
- **Mandatory headers**:
  - Content-Type: 'application/json',
  - Connection: 'keep-alive',
- **Payload**:

Field	Description
authentication	Valid authentication token
private_key	The user's private key encrypted in PEM format, and base64-encoded
perimeter_id	Optional string
supported_algorithms	List of supported algorithms:['RSA/ECB/PKCS1Padding', 'RSA/ECB/OAEPwithSHA-1andMGF1Padding', 'RSA/ECB/OAEPwithSHA-256andMGF1Padding', 'RSA/ECB/OAEPwithSHA-512andMGF1Padding' 'SHA1withRSA', 'SHA256withRSA', 'SHA512withRSA', 'SHA1withRSA/PSS', 'SHA256withRSA/PSS', 'SHA512withRSA/PSS'];

EXAMPLE:
Request enabling the encryption of a private key, sent in POST over the */wrapprivatekey* route:

```
{
  "authentication": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImFjZGEz...",
  "private_key": "LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVk...",
  "supported_algorithms": [
        "RSA/ECB/PKCS1Padding",
        "RSA/ECB/OAEPwithSHA-1andMGF1Padding",
        "RSA/ECB/OAEPwithSHA-256andMGF1Padding",
        "RSA/ECB/OAEPwithSHA-512andMGF1Padding,"
        "SHA1withRSA",
        "SHA256withRSA",
        "SHA512withRSA"
        "SHA1withRSA/PSS",
        "SHA256withRSA/PSS",
        "SHA512withRSA/PSS"
  ]
}
```

The response to this request is a JSON object named *wrapped_private_key* which contains a string representing the encrypted private key.

Providing private keys to Google

- Enable Gmail and provide your users' encrypted private keys and certification chains. For more information, refer to the Google documentation Gmail only: Set up your organization for client-side encryption.
  Certification chains must meet the following Google specifications:
  - S/MIME certificate profiles,
  - Set up rules to require S/MIME.

Using Gmail

- To use Gmail to send encrypted messages to internal or external users, refer to Google documentation Learn about Gmail Client-side encryption.

10.10.2 Using Gmail in advanced mode based on a KMS

Configuring the KACLS

Modify the *config.json* file as described in the steps below: For more information, refer to the Configuring the KACLS section.

1. Fill in the whole `crypto_backend` section, and assign the "kms" value to the `type` field. See crypto_backends parameter and Configuration of the cryptographic backend.

   > **NOTE**
   > If using a KMS domain, the *domain_id* values of all the tenant configurations must imperatively be the same and match the domain used by the KMIP configuration.

2. In the `id` field of the `keys` section, enter the UUID of the cryptographic backend set in step 1. See keys parameters and Configuration of the cryptographic backend.
3. Fill in the `authorization` section with Gmail information as shown in Authorization settings.
4. Fill in the `wrapprivatekey_authentication` section as shown in wrapprivatekey_ authentication parameters.

5. Optional. Fill in the `admin_authentication` section as shown in admin_authentication parameters to perform privileged operations.

Encrypting users' private keys with the KACLS

Ensure that the Stormshield KMaaS is fully configured and operational before following the steps below.

- For every private key to be encrypted, send a POST request to the */wrapprivatekey* API route with the following headers and payload:
- **URL**: in the format "{protocol: http | https}://{kacls url}/api/v1/{tenantId}/wrapprivatekey", where:
  - {kacls url} is the URL of the key service that you have declared in Specifying the External key service
  - {tenantId} is your tenant's UUID.

- **Mandatory headers**:
  - Content-Type: 'application/json',
  - Connection: 'keep-alive',
- **Payload**:

Field	Description
authentication	Valid administrator authentication token
private_key	ID of the user's private key stored in the KMS, and base64-encoded.
perimeter_id	Optional string
supported_algorithms	List of supported algorithms: ['RSA/ECB/PKCS1Padding', 'SHA1withRSA', 'SHA256withRSA', 'SHA512withRSA', 'SHA1withRSA/PSS', 'SHA256withRSA/PSS', 'SHA512withRSA/PSS'];
public_key	Public key of the user in PEM format, and base64-encoded.

EXAMPLE:
Request enabling the encryption of a private key, sent in POST over the */wrapprivatekey* route:

```
{
  "authentication": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImFjZGEz...",
  "private_key": "LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVk...",
  "supported_algorithms": [
        "RSA/ECB/PKCS1Padding",
        "SHA1withRSA/PSS",
        "SHA256withRSA/PSS",
        "SHA512withRSA",
        "SHA512withRSA/PSS"
  ],
  "public_key" : "e32tLS1CRUdJTiBSU0FgUFJJVkFURSBLRck..."
}
```

- For every private key to be encrypted, call up the */wrapprivatekey* API route in POST with the following headers and payload:
- **URL**: in the format "{protocol: http | https}://{kacls url}/api/v1/{tenantId}/wrapprivatekey", where:
  - {kacls url} is the URL of the key service that you have declared in Specifying the External key service
  - {tenantId} is your tenant's UUID.

- **Mandatory headers**:
  - Content-Type: 'application/json',
  - Connection: 'keep-alive',
- **Payload**:

Field	Description
authentication	Valid administrator authentication token
private_key	ID of the user's private key stored in the KMS, and base64-encoded.
perimeter_id	Optional string
supported_algorithms	List of supported algorithms: ['RSA/ECB/PKCS1Padding', 'SHA1withRSA', 'SHA256withRSA', 'SHA512withRSA', 'SHA1withRSA/PSS', 'SHA256withRSA/PSS', 'SHA512withRSA/PSS'];
public_key	Public key of the user in PEM format, and base64-encoded.

EXAMPLE:
Request enabling the encryption of a private key, sent in POST over the */wrapprivatekey* route:

```
{
  "authentication": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImFjZGEz...",
  "private_key": "LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVk...",
  "supported_algorithms": [
        "RSA/ECB/PKCS1Padding",
        "SHA1withRSA/PSS",
        "SHA256withRSA/PSS",
        "SHA512withRSA",
        "SHA512withRSA/PSS"
  ],
  "public_key" : "e32tLS1CRUdJTiBSU0FgUFJJVkFURSBLRck..."
}
```

Providing the encrypted ID of the private keys to Google

- Enable Gmail and provide your user's private key encrypted IDs and certification chains. For more information, refer to the Google documentation Gmail only: Set up your organization for client-side encryption.
  Certification chains must meet the following Google specifications:
  - S/MIME certificate profiles,
  - Set up rules to require S/MIME signature and encryption

10.10.3 Using Gmail

- To use Gmail to send encrypted messages to internal or external users, refer to Google documentation Learn about Gmail Client-side encryption.

10.10.4 Using Gmail in Send to Anyone mode

If you have the Google Workspace Assured Controls license, you can use Gmail Send to Anyone feature (STA) to send encrypted emails both to internal and external users, without having to configure S/MIME certificates.

To use STA with users outside the company, enable Guest access by following the procedure described in section Enabling external user access for Google Drive and Google Meet. In the **Configure guest IdP** section of the Google Admin, select the **Gmail** option.

To configure the STA mode:

- In the *config.json* file, fill in the *authorization* section with Gmail STA information as described in section authorization parameters.

For more information, refer to Google's documentation:

- About Assured Controls and Assured Controls Plus
- Learn about Gmail Client-side encryption

10.11 Migrating an external key service to another

If you have an external third-party key service (also known as a KACLS) and you want to replace it with the KACLS, follow the Google migration procedure. During this procedure, you will be able to retrieve all your old encrypted data and re-encrypt it to the KACLS.

Before launching the migration, you must choose a backup key service to which old data will also be encrypted. Google will launch two parallel migrations: encrypted data will be migrated to the KACLS and to the backup key service.

The KACLS must be configured before migration is enabled in the Google Admin interface.

10.11.1 Configuring migration in the KACLS

1. Generate a pair of RSA keys without password in PEM format with the tool of your choice. For example with OpenSSL and the following commands:
```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_
keygen_bits:4096
openssl rsa -pubout -in private_key.pem -out public_key.pem
```
2. Encode the generated private key in base64 with the following command:
```
openssl base64 -A -in private_key.pem -out private_key_base64.txt
```

3. Fill in the `"migration"` section in the *config.json* file:
   - In the `"format"` field, specify the PEM key format,
   - In the `"key"` field, enter the private key generated in base64,
   - In the `"kacls_urls"` field, add all the key services with which the Stormshield KMaaS must communicate, including the backup key service.

```
"migration": {
    "enabled": true,
    "kaclstokacls_token": {
        "kid": "key_identifier",
        "format": "pem",
        "key": "a_private_key",
        "duration": 3600
    },
    "acls": {
        "kacls_urls": ["https://kacls_1", "https://kacls_2"]
    }
   }
```

   For more information, refer to Configuring the KACLS, in particular the section on Migration.
4. In the *config.json* file, fill in the `authorization` section with information on the migration, as shown in authorization parameters.

10.11.2 Adding the KACLS in Google

1. In the Google Workspace administration console, add the KACLS as a new key service
2. Select an operational backup key service as well, to which old data will also be encrypted. This backup key service can be the one that you wish to replace.

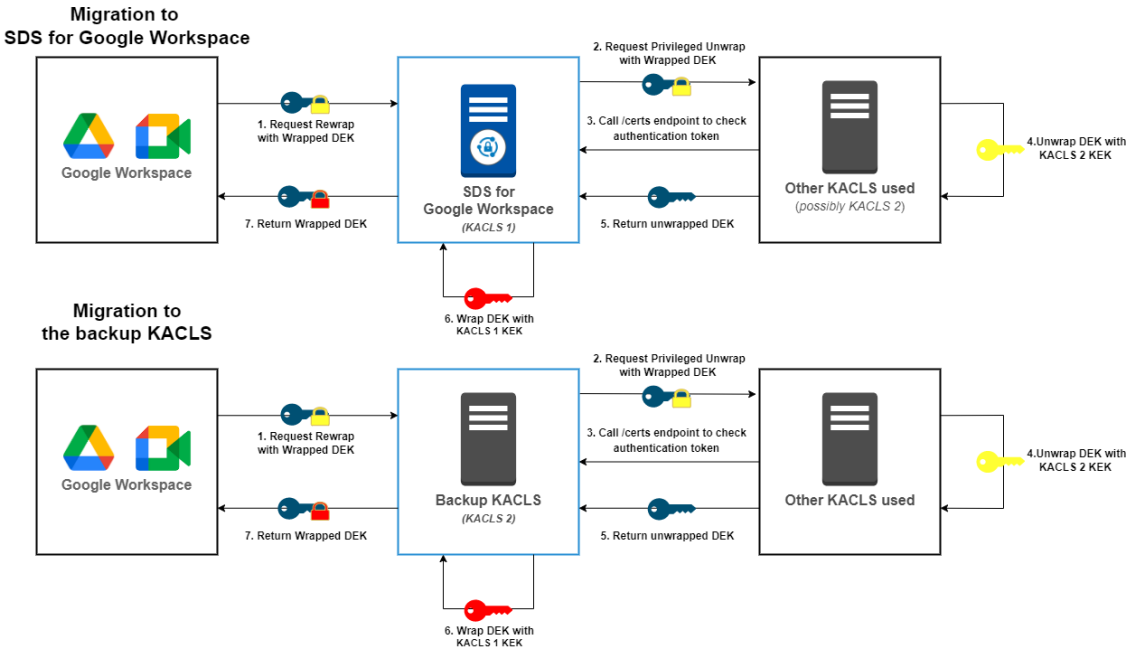
For more information, refer to the Google documentation.

10.11.3 Enabling key service migration in Google

1. In the Google Workspace administration console, select the KACLS key service.
2. Enable key service migration.
   Google will launch the migration. Refer to the corresponding logs in the KACLS. See the *Log Guide*.

For more information, refer to the Google documentation.

The diagram below shows the various stages of the migration to the KACLS and to the backup key service. In this diagram, the term 'KACLS' refers to a key service.



NOTE
The tokens generated by the KACLS and provided to another KACLS in the Privileged Unwrap request are signed using the RS256 algorithm.

10.11.4 Using the backup key service other than for migration service

The backup key service guarantees access to content if an issue arises with the main key service. It is mandatory in a migration, but you can also use it to test a new key service, for example. In this case, encrypting data to the backup key service is still considered a migration, and you must configure the KACLS accordingly:

- Fill in the `"migration"` section in the *config.json* file. In the `"kacls_urls"` field, add the backup key service.

```
"migration": {
    "enabled": true,
    "kaclstokacls_token": {
        "kid": "key_identifier",
        "format": "pem",
        "key": "a_private_key",
        "duration": 3600
    },
    "acls": {
        "kacls_urls": ["https://backup_kacls"]
    }
   }
```

10.12 Customizing the authorization rules

You can customize the rules that allow or deny a request to the Stormshield KMaaS, using Open Policy Agent (OPA). The policy evaluates the request inputs. If the request is forbidden, the access is denied and the "403 Forbidden" error is returned.

In the *config.json* file, the *policy_enforcement.enable* parameter is mandatory. It allows you to specify whether you want to enable OPA rules or not.

The inputs relating to all API routes are described in section Inputs relating to all API routes.

The tables below describe the inputs specific to the KACLS.

10.12.1 Inputs relating to all KACLS routes

These inputs are meant to create customized rules to access the KACLS. Use them to filter the application of policies.

You can use these inputs in the custom policy.

Input	Description	Source of the input
perimeterId	Optional. Value relating to the key used to wrap a DEK.	Data encapsulated in the DEK within the request body
resourceName	Unique identifier of the object encrypted by the DEK.	Data encapsulated in the DEK within the request body
reason	String containing a JSON object. Not currently used.	Request body

10.12.2 Inputs specific to the wrap and unwrap API routes

Input	Description	Source of the input
authentication.email	User's email address. In UTF8 format, in lower case.	JWT authentication token provided by the IDP.
authentication.googleEmail	Optional If the email field is not a Google Workspace address belonging to the domain, googleEmail is the user's email address. It takes priority over the email field. In UTF8 format, in lower case.	JWT authentication token provided by the IDP.
authentication.iss	Entity which has created and signed the token.	JWT authentication token provided by the IDP.
authentication.aud	Corresponds to the audience for which the token was issued. Example: ['cse-authorization', 'cse-authorization1']	JWT authentication token provided by the IDP.
authentication.iat	Date when the token was issued. In timestamp format (integer) Example: 1677679386	JWT authentication token provided by the IDP.

Input	Description	Source of the input
authentication.exp	Date when the token expires. In timestamp format (integer) Example: 1677679386	JWT authentication token provided by the IDP.
authentication.customClaims	Optional. Custom claims provided par the IDP. See Inputs specific to the wrap and unwrap API routes.	JWT authentication token provided by the IDP.
authorization.iss	Entity which has created and signed the token. Can be used to differentiate the various Google Workspace applications, or to migrate from a KACLS to another. Example: “gsuitecse-tokenissuer-drive@system.gserviceaccount.com” See Authorization settings.	JWT authorization token provided by Google
authorization.role	Role of the authorized user. • "writer" for wrap and unwrap, • "reader" for unwrap	JWT authorization token provided by Google
authorization.aud	Corresponds to the audience for which the token was issued. Example: ['cse-authorization', 'cse-authorization1']	JWT authorization token provided by Google
authorization.exp	Date when the token expires. In timestamp format (integer) Example: 1677679386	JWT authorization token provided by Google
authorization.iat	IssuedAt, date when the token was issued. In timestamp format (integer) Example: 1677679386	JWT authorization token provided by Google
authorization.emailType	Identifies the origin of the e-mail address in the token. Prescribed values: • "google" for Google accounts (default value), • "google-visitor" for Google-verified accounts, • "customer-idp" for IDP accounts.	JWT authorization token provided by Google
authorization.kaclsOwnerDomain	Optional claim configured by the Google Workspace's owner to change the domain used in the request to the KACLS. It prevents from registering unauthorized KACLS. This claim is sent to the OPA server, which allows or forbids the action based on this information.	JWT authorization token provided by Google
contentType	Cryptographic content type. Prescribed value: "dek"	Cryptographic component used.

10.12.3 Inputs specific to the privilegedwrap and privilegedunwrap API routes

Input	Description	Source of the input
authentication.email	User's email address. In UTF8 format, in lower case.	JWT authentication token provided by the IDP.
authentication.googleEmail	Optional If the email field is not a Google Workspace address belonging to the domain, googleEmail is the user's email address. It takes priority over the email field. In UTF8 format, in lower case.	JWT authentication token provided by the IDP.
authentication.iss	Entity which has created and signed the token.	JWT authentication token provided by the IDP.
authentication.aud	Corresponds to the audience for which the token was issued. Example: ['cse-authorization', 'cse-authorization1']	JWT authentication token provided by the IDP.
authentication.iat	Date when the token was issued. In timestamp format (integer) Example: 1677679386	JWT authentication token provided by the IDP.
authentication.exp	Date when the token expires. In timestamp format (integer) Example: 1677679386	JWT authentication token provided by the IDP.
authentication.customClaims	Optional. Custom claims provided par the IDP. SeeInputs specific to the privilegedwrap and privilegedunwrap API routes.	JWT authentication token provided by the IDP.
contentType	Cryptographic content type. Prescribed value: "dek"	Cryptographic component used.

10.12.4 Inputs specific to the rewrap API route

Input	Description	Source of the input
originalKaclsUrl	URL of the initial KACLS for a migration.	Request body
authorization.email	User's email address. In UTF8 format, in lower case.	JWT authorization token provided by Google
authorization.role	Role of the authorized user: "migrator"	JWT authorization token provided by Google
authorization.resourceName	Same as resourceName, but originates from the authorization token.	JWT authorization token provided by Google

Input	Description	Source of the input
authorization.kaclsUrl	URL of the KACLS.	JWT authorization token provided by Google
authorization.iss	Entity which has created and signed the token. Can be used to differentiate the various Google Workspace applications, or to migrate from a KACLS to another. Example: “gsuitecse-tokenissuer-drive@system.gserviceaccount.com” See Authorization settings.	JWT authorization token provided by Google
authorization.aud	Corresponds to the audience for which the token was issued. Example: ['cse-authorization', 'cse-authorization1']	JWT authorization token provided by Google
authorization.exp	Date when the token expires. In timestamp format (integer) Example: 1677679386	JWT authorization token provided by Google
authorization.iat	IssuedAt, date when the token was issued. In timestamp format (integer) Example: 1677679386	JWT authorization token provided by Google
authorization.kaclsOwnerDomain	Optional claim configured by the Google Workspace's owner to change the domain used in the request to the KACLS. It prevents from registering unauthorized KACLS. This claim is sent to the OPA server, which allows or forbids the action based on this information.	JWT authorization token provided by Google

10.12.5 Inputs specific to the certs API route

Only *endpoint* and *tenantId* inputs are available for the *certs* API route. For more information, refer to the section Inputs specific to the certs API route.

10.12.6 Inputs specific to the digest and systemwrap API routes

Input	Description	Source of the input
authorization.email	User’s email address. In UTF8 format, in lower case.	JWT authorization token provided by Google
authorization.role	Role of the authorized user: "check"	JWT authorization token provided by Google
authorization.resourceName	Same as resourceName, but originates from the authorization token.	JWT authorization token provided by Google

Input	Description	Source of the input
authorization.kaclsUrl	URL of the KACLS.	JWT authorization token provided by Google
authorization.iss	Entity which has created and signed the token. Can be used to differentiate the various Google Workspace applications, or to migrate from a KACLS to another. Example: “gsuitecse-tokenissuer-drive@system.gserviceaccount.com” See Authorization settings.	JWT authorization token provided by Google
authorization.aud	Corresponds to the audience for which the token was issued. Example: ['cse-authorization', 'cse-authorization1']	JWT authorization token provided by Google
authorization.exp	Date when the token expires. In timestamp format (integer) Example: 1677679386	JWT authorization token provided by Google
authorization.iat	IssuedAt, date when the token was issued. In timestamp format (integer) Example: 1677679386	JWT authorization token provided by Google
authorization.kaclsOwnerDomain	Optional claim configured by the Google Workspace's owner to change the domain used in the request to the KACLS. It prevents from registering unauthorized KACLS. This claim is sent to the OPA server, which allows or forbids the action based on this information.	JWT authorization token provided by Google

10.12.7 Inputs specific to the privatekeydecrypt and privatekeysign API routes

Input	Description	Source of the input
algorithm	Algorithm used to encrypt the private key.	Data within the request body.
authentication.email	User's email address. In UTF8 format, in lower case.	JWT authentication token provided by the IDP.
authentication.googleEmail	Optional If the email field is not a Google Workspace address belonging to the domain, googleEmail is the user's email address. It takes priority over the email field. In UTF8 format, in lower case.	JWT authentication token provided by the IDP.
authentication.iss	Entity which has created and signed the token.	JWT authentication token provided by the IDP.

Input	Description	Source of the input
authentication.aud	Corresponds to the audience for which the token was issued. Example: ['cse-authorization', 'cse-authorization1']	JWT authentication token provided by the IDP.
authentication.iat	Date when the token was issued. In timestamp format (integer) Example: 1677679386	JWT authentication token provided by the IDP.
authentication.exp	Date when the token expires. In timestamp format (integer) Example: 1677679386	JWT authentication token provided by the IDP.
authentication.customClaims	Optional. Custom claims provided by the IDP. See Inputs specific to the privatekeydecrypt and privatekeysign API routes.	JWT authentication token provided by the IDP.
authorization.email	User's email address. In UTF8 format, in lower case.	JWT authorization token provided by Google
authorization.role	Role of the authorized user. • "decrypter" for privatekeydecrypt, • "signer" for privatekeysign	JWT authorization token provided by Google
authorization.resourceName	Same as resourceName, but originates from the authorization token.	JWT authorization token provided by Google
authorization.perimeterId	Same as perimeterId, but originates from the authorization token.	JWT authorization token provided by Google
authorization.kaclsUrl	URL of the KACLS.	JWT authorization token provided by Google
authorization.iss	Entity which has created and signed the token. Can be used to differentiate the various Google Workspace applications, or to migrate from a KACLS to another. Example: "gsuitecse-tokenissuer-drive@system.gserviceaccount.com" See Authorization settings.	JWT authorization token provided by Google
authorization.aud	Corresponds to the audience for which the token was issued. Example: ['cse-authorization', 'cse-authorization1']	JWT authorization token provided by Google

Input	Description	Source of the input
authorization.exp	Date when the token expires. In timestamp format (integer) Example: 1677679386	JWT authorization token provided by Google
authorization.iat	IssuedAt, date when the token was issued. In timestamp format (integer) Example: 1677679386	JWT authorization token provided by Google
authorization.spkiHashBase64	SPKI hash in base64 to validate authorization.	JWT authorization token provided by Google
authorization.spkiHashAlgorithm	Encryption algorithm used to produce the SPKI hash.	JWT authorization token provided by Google
authorization.messageId	Optional. Value relating to the encryption key used during a wrapprivatekey operation.	JWT authorization token provided by Google
authorization.kaclsOwnerDomain	Optional claim configured by the Google Workspace's owner to change the domain used in the request to the KACLS. It prevents from registering unauthorized KACLS. This claim is sent to the OPA server, which allows or forbids the action based on this information.	JWT authorization token provided by Google
contentType	Cryptographic content type. Prescribed values: "private-key-pem" or "private-key-name"	Cryptographic component configured.

10.12.8 Inputs specific to the wrapprivatekey and privilegedprivatekeydecrypt API routes

Input	Description	Source of the input
authentication.email	User's email address. In UTF8 format, in lower case.	JWT authentication token provided by the IDP.
authentication.googleEmail	Optional if the email field is not a Google Workspace address belonging to the domain, googleEmail is the user's email address. It takes priority over the email field. In UTF8 format, in lower case.	JWT authentication token provided by the IDP.
authentication.iss	Entity which has created and signed the token.	JWT authentication token provided by the IDP.
authentication.aud	Corresponds to the audience for which the token was issued. Example: ['cse-authorization', 'cse-authorization1']	JWT authentication token provided by the IDP.

Input	Description	Source of the input
authentication.iat	Date when the token was issued. In timestamp format (integer) Example: 1677679386	JWT authentication token provided by the IDP.
authentication.exp	Date when the token expires. In timestamp format (integer) Example: 1677679386	JWT authentication token provided by the IDP.
authentication.customClaims	Optional. Custom claims provided par the IDP. SeeInputs specific to the wrapprivatekey and privilegedprivatekeydecrypt API routes.	JWT authentication token provided by the IDP.
contentType	Cryptographic content type. Prescribed values: "private-key-pem" or "private-key-name"	Cryptographic component configured.

11. Double Key Encryption (DKE)

The DKE module allows you to configure your backend server to protect your data in Microsoft environment. For more information, see the Microsoft documentation.

This module exposes:

- A public key retrieval endpoint used by Microsoft to get the keys used during local encryption of Data Encryption Keys (DEKs),
- A decryption endpoint used by Microsoft to request decryption of the previously encrypted DEKs.

All cryptographic material remains under the exclusive control of the Stormshield Stormshield KMaaS.

11.1 Understanding the requirements

- The DKE module is fully compatible with Windows and iOS.
  Decryption works on macOS and Android, but encryption has not yet been tested.
- You must generate your KEKs and add them to the *keks.json* file, in the *dke_keys* section:
  For more information, see Generating KEKs.
- The DKE module is not compatible with a KMS. It will not be enabled if the Stormshield KMaaS is started with the *persistence_type* set to "kms".
- Microsoft Purview sensitivity labels must be configured to reference the public key endpoint exposed by the Stormshield KMaaS.
- For information on global requirements and recommendations, refer to section Understanding the global requirements

11.2 Configuring the DKE

The DKE module is configured in the *dke* section of the *config.json* file. You can configure it independently for each tenant.

The template for the DKE configuration block is as follows:

```
"dke": {
  "enable": true,
  "directory_tenant_id": "_DIRECTORY_TENANT_ID_",
  "authentication": [
    {
        "discovery_uri": "_DKE_DISCOVERY_URI_",
        "client_id": "_DKE_CLIENT_ID_"
    }
  ],
  "policy_enforcement": {
     "enable": true,
     "type": "_POLICY_ENFORCEMENT_TYPE_",
     "opa_server": {
       "url": "_URL_",
       "authentication": {
         "type": "basic",
         "user_id": "_USER_ID_",
         "password": "_PASSWORD_"
       }
     }
   },
   "cache": {
```

```
        "enable": false,
        "duration_in_seconds": "_DKE_CACHE_DURATION_IN_SECONDS_"
      }
    }
```

	Description	Type	Optional/ mandatory
enable	Enables or disables the DKE module for the tenant.	Boolean	Mandatory to use the DKE module
directory_tenant_id	ID of tenant in Azure Purview.	String in uuid format	Mandatory to use the DKE module

Authentication parameter

JSON object containing the configuration that allows authenticating to the DKE module.

Identity provisioning is managed by Azure Purview services.

Parameter	Description	Type	Optional/ mandatory
discovery_uri	URL to the OpenID JSON configuration file for OpenID authentication. The token is issued by Microsoft Azure. The discovery_uri must match the issuer of the JWT sent: *https://sts.windows.net/<YOUR_DIRECTORY_TENANT_ID>/.well-known/openid-configuration* .	String	Mandatory
client_id	Recipient of the JWT authentication token (see RFC 7519). An entry must be added for each identity provider. The client_id must match the audience of the JWT sent: *<BASE_URL>* The *BASE_URL* must match the *kacls_url* configuration and the Application ID URI of your Azure application used for DKE.	String	Mandatory

policy_enforcement parameter

JSON object containing the configuration of the optional OPA enforcement feature for the DKE module. For more information, see Implementing the authorization rules with Open Policy Agent.

Parameter			
enable	Enable the use of OPA rules for the module.	Boolean	Mandatory for each module enabled except the PKI.

Parameter			
type	Kind of OPA policy to use. The possible values are: • opa_local: this mode uses local files *policy.wasm* and *policy.data.json*. The file names must be adapted according to the module used. • opa_server: this mode uses a remote OPA server.	String	Mandatory if policy_enforcement.enable is set to true
opa_server: JSON object describing the parameters required to access the OPA policy server. Stormshield guarantees compatibility with OPA version 1.2.0.			
url	URL of data API exposed endpoints. For more information, see OPA documentation. *Example*: If your rego package is *stormshield.kmaas* and you have the *allow* variable in this package, your url will be: *https://opa-server/v1/data/stormshield/kmaas/allow* The authorized protocols are http and https. Stormshield strongly recommends https in production.	String	Mandatory if policy_enforcement.type is set to opa_server
authentication	JSON object describing the parameters required to authenticate to the OPA policy server. It includes the following fields: • type: Type of authentication used to connect to the policy server. The prescribed value is "basic". • user_id: Identifier of the user account used to connect to the policy server. Mandatory if authentication.type is set on "basic" • password: Password of the user account used to connect to the policy server. Mandatory if authentication.type is set on "basic"	Object	Mandatory if type is set to opa_server

EXAMPLE
The content below is provided as an example and must not be used as such in the Stormshield KMaaS DKE configuration of one of your tenants.

cache parameter

JSON object containing the configuration that controls the cache for the *getKey* route of the DKE module.

Parameter	Description	Type	Optional/ mandatory
enable	Enable or disable the cache in *getKey* response.	Boolean	Mandatory
duration_in_seconds	Specify how long a public key remains valid after a *getKey* request. Its expiration date is indicated by the *cache.exp* parameter in the response. For example, if *duration_in_seconds* is 3600 (one hour), then *cache.exp* will be the date corresponding to one hour after the request is made.	Number (positive integer)	Mandatory if enable is set to true

11.3 Configuring Microsoft Purview and Azure

To use the DKE module, you must create an application in Azure, then a Microsoft Purview sensitivity label in the Microsoft administration.

11.3.1 11.3.1 Creating the application in Azure

To enable Double-Key Encryption (DKE), you first need to register a new application in Azure. For more information, follow the procedure in the Microsoft documentation.

When you reach the **Expose an API** section, next to **Application ID URI**, enter the *BASE_URL* (i.e., the value of the *kacls_url* parameter) that you specified in the *authentication* section. For more information, see the authentication table.

11.3.2 Creating a sensitivity label with DKE encryption

The sensitivity label determines which content should be protected using Double Key Encryption. You can give it any name of you choice, as it does not correspond directly to any Stormshield KMaaS parameter.

1. From the Microsoft Purview portal, create a new sensitivity label.
2. In the **Encryption** panel, check the **Use Double Key Encryption** option.
3. Enter the public key endpoint URL exposed by the Stormshield KMaaS: */api/v1/{tenantid}/dke/{keyId}*.

This endpoint must return a public RSA key in JWK format.

For more information, refer to the Microsoft documentation.

11.4 Customizing the authorization rules

You can customize the rules that allow or deny a request to the Stormshield KMaaS, using Open Policy Agent (OPA). The policy evaluates the request inputs. If the request is forbidden, the access is denied and the "403 Forbidden" error is returned.

In the *config.json* file, the *policy_enforcement.enable* parameter is mandatory. It allows you to specify whether you want to enable OPA rules or not.

The inputs relating to all API routes are described in section Inputs relating to all API routes.

The tables below describe the inputs specific to the DKE module.

11.4.1 Inputs specific to the DKE get and decrypt API routes

You can use the following inputs to create custom rules to access the DKE module, for example to filter the application of policies.

Input	Description	Source of the input
kid	URI identifying the exact key version.	URL of the request.
versionId	Identifier of the version of the key used for the request.	URL of the request.
userAgent	Identifier of the application issuing the HTTP request.	Header of the HTTP request.

11.4.2 Inputs specific to the DKE decrypt API route

You can use the following inputs to create custom rules to access the DKE module on the decrypt route.

Input	Description	Source of the input
authentication.iss	Entity which has created and signed the token.	JWT authentication token provided by the IDP.
authentication.aud	Corresponds to the audience for which the token was issued. Example: https://cse.mysds.io	JWT authentication token provided by the IDP.
authentication.iat	Date when the token was issued. In timestamp format (integer) Example: 1677679386	JWT authentication token provided by the IDP.
authentication.exp	Date when the token expires. In timestamp format (integer) Example: 1677679386	JWT authentication token provided by the IDP.
authentication.customClaims	Custom claims provided par the IDP. See the List of claims provided by Microsoft Azure services. All payload claims, expect *iss*, *aud*, *iat* and *exp* may be present in custom claims.	JWT authentication token provided by the IDP.

11.5 Using the DKE module

For each DKE key defined by its *kid*, the Stormshield KMaaS may store multiple key versions (i.e., *version_id*):

- The most recent key version is automatically selected and used for encryption,
- Older key versions are retained to allow decryption of previously encrypted DEKs,
- Microsoft identifies a key version through its *version_id* when requesting decryption,
- Key rotation is transparent for Microsoft and does not impact access to existing data.

11.5.1 Retrieving keys

When Microsoft needs to encrypt a DEK, it calls the Stormshield KMaaS *GET /api/v1/{tenantid}/dke/{kid}* route associated with the sensitivity label. The Stormshield KMaaS returns the most recent version of the public key in JWK format.

The *kid* field in the response is a URI identifying the exact key version which will be later reused by Microsoft for decryption requests.

For more information, see the API documentation.

11.5.2 Decrypting DEKs

When Microsoft needs to decrypt a DEK, the workflow is the following:

1. Microsoft calls the Stormshield KMaaS *POST /api/v1/{tenantid}/dke/{kid}/{versionId}/Decrypt* route.
2. The Stormshield KMaaS resolves the requested key version.
3. It uses the associated private key to decrypt the encrypted DEK,
4. It returns the plaintext DEK to Microsoft.

If the key version does not exist or is disabled, the request issues an error.

For more information, see the API documentation.

12. Crypto API

Crypto API provides two API routes, *crypto/encrypt* and *crypto/decrypt*. These routes are designed for general-purpose cryptographic operations, independent of any specific ecosystem

Before performing cryptographic operations, Stormshield KMaaS first checks the authentication used. It can be either:

- User authentication with a JWT token,
- API_key with a previously declared key.

Crypto API generates logs for all the operations that it performs.

For more information on API routes, see the API documentation.

NOTE
The use of the solution in any way other than as described in the documentation is not managed. Alternatively, get in touch with Stormshield Support for clarification.

12.1 Understanding the requirements

- You must generate your KEKs and add them to the *keks.json* file. For more information, see Configuring KEKs.
- For information on global requirements and recommendations, refer to section Understanding the global requirements

12.2 Configuring Crypto API

Crypto API is configured in the *crypto_api* section of the *config.json* file. You can configure it independently for each tenant. For more information on a tenant global configuration, see Configuring the Stormshield KMaaS.

The template for the *crypto_api* configuration block is as follows:

```
"crypto_api": {
  "enable": _IS_CRYPTO_API_ENABLE_,
  "authentication": [
   {
    "discovery_uri": "_CRYPTO_API_AUTHENTICATION_DISCOVERY_URI_",
    "client_id": "_CRYPTO_API_AUTHENTICATION_ISSUER_"
   },
   {
    "name": "_CRYPTO_API_AUTHENTICATION_API_KEY_NAME",
    "api_key": "_CRYPTO_API_AUTHENTICATION_API_KEY_VALUE"
   }
  ],
  "policy_enforcement": {
   "enable": false,
   "type": "_POLICY_ENFORCEMENT_TYPE_",
   "opa_server": {
    "url": "_URL_",
    "authentication": {
     "type": "basic",
     "user_id": "_USER_ID_",
     "password": "_PASSWORD_"
```

```
        }
      }
    }
  },
}
```

Parameter	Description	Type	Optional/ mandatory
enable	Enables or disables the CryptoAPI module, Crypto API.	Boolean	Mandatory to use the CryptoAPI module.

Authentication parameter

JSON object containing the configuration that allows authenticating to Crypto API. There are two types of authentication: with OpenID and JWT tokens or API keys.

When using the first method, you must have configured the Identity Provider (IDP) to deliver JWT tokens with at least these fields: "iss", "aud", "exp", "iat". For more information, see Configuring the identity provider.

Parameter	Description	Type	Optional/ mandatory
discovery_uri	URL to the OpenID JSON configuration file for the OpenID authentication.	String	Mandatory to use OpenID authentication.
client_id	Recipient of the JWT authentication token (see RFC 7519). An entry must be added for each identity provider.	String	Mandatory to use OpenID authentication.
name	Name of the API key. The character ":" is not allowed. See RFC 2617.	String	Mandatory to use API Key authentication.
api_key	Value of the API key. Must be a valid API key provided by Stormshield.	String	Mandatory to use API Key authentication.

policy_enforcement parameter

JSON object containing the configuration of the optional OPA enforcement feature for Crypto API. For more information, see Implementing the authorization rules with Open Policy Agent.

Parameter			
enable	Enable the use of OPA rules for the module.	Boolean	Mandatory for each module enabled except the PKI.

Parameter			
type	Kind of OPA policy to use. The possible values are: • opa_local: this mode uses local files *policy.wasm* and *policy.data.json*. The file names must be adapted according to the module used. • opa_server: this mode uses a remote OPA server.	String	Mandatory if policy_enforcement.enable is set to true
opa_server: JSON object describing the parameters required to access the OPA policy server. Stormshield guarantees compatibility with OPA version 1.2.0.			
url	URL of data API exposed endpoints. For more information, see OPA documentation. *Example*: If your rego package is *stormshield.kmaas* and you have the *allow* variable in this package, your url will be: *https://opa-server/v1/data/stormshield/kmaas/allow* The authorized protocols are http and https. Stormshield strongly recommends https in production.	String	Mandatory if policy_enforcement.type is set to opa_server
authentication	JSON object describing the parameters required to authenticate to the OPA policy server. It includes the following fields: • type: Type of authentication used to connect to the policy server. The prescribed value is "basic". • user_id: Identifier of the user account used to connect to the policy server. Mandatory if authentication.type is set on "basic" • password: Password of the user account used to connect to the policy server. Mandatory if authentication.type is set on "basic"	Object	Mandatory if type is set to opa_server

12.3 Customizing the authorization rules

You can customize the rules that allow or deny a request to the Stormshield KMaaS, using Open Policy Agent (OPA). The policy evaluates the request inputs. If the request is forbidden, the access is denied and the "403 Forbidden" error is returned.

In the *config.json* file, the *policy_enforcement.enable* parameter is mandatory. It allows you to specify whether you want to enable OPA rules or not.

The inputs relating to all API routes are described in section Inputs relating to all API routes.

The tables below describe the inputs specific to the Crypto API.

12.3.1 Inputs specific to Crypto API encrypt and decrypt routes

As a Stormshield KMaaS administrator, you can use these inputs to create custom rules to access Crypto API, for example to filter the application of policies. Use some of these inputs in the created policy.

Input	Description	Source of the input
endpoint	API routes called: "crypto/encrypt", "crypto/decrypt"	URL of the request
tenantId	unique identifier of a tenant in UUID format. Example: 2363615f-5b08-4119-a5bd-fad3f5f3f420	URL of the request
authentication.authType	Type of authentication used for the request. Possible values : • token, • apiKey	"authorization" header used.
authentication.keyName	Name of the API key used to authenticate the request. Only present when using "apiKey" authentication.	Name corresponding to the key in configuration.
authentication.iss	Entity which has created and signed the token. Only present when using "token" authentication.	JWT authentication token provided by the IDP.
authentication.aud	Corresponds to the audience for which the token was issued. Only present when using "token" authentication. Example: ['cse-authorization', 'cse-authorization1']	JWT authentication token provided by the IDP.
authentication.iat	Date when the token was issued. In timestamp format (integer) Only present when using "token" authentication. Example: 1677679386	JWT authentication token provided by the IDP.
authentication.exp	Date when the token expires. In timestamp format (integer) Only present when using "token" authentication. Example: 1677679386	JWT authentication token provided by the IDP.
authentication.customClaims	Optional. Custom claims provided par the IDP. Only present when using "token" authentication. See Inputs specific to Crypto API encrypt and decrypt routes.	JWT authentication token provided by the IDP.

Inputs specific to Crypto API decrypt route

Input	Description	Source of the input
kekId	Identifier of the KEK used for data decryption.	Body of the decrypt request, or ID of the active KEK if not specified.

Input	Description	Source of the input
policy	Content of the optional policy block included in the decrypt request body. You can use it to define ABAC policies using *policy.body.dataAttributes*. It is present only when the policy block is included in the decrypt request or when using Stormshield SDK.	Body of the decrypt request.

13. Key Access Service (KAS)

The Key Access Service is a dedicated backend component that manages the cryptographic keys used for encryption workflows based on Trusted Data Format (TDF). It acts as a secure intermediary (i.e., key server) that stores, manages, and provides controlled access to symmetric and asymmetric Key Encryption Keys (KEKs).

The Key Access Service is required to use Stormshield SDK.

It provides the following API routes:

- */rewrap*: Enables clients to securely retrieve or re-encrypt keys needed to decrypt protected data, while enforcing policies and access controls (i.e., asymmetric cryptography).
- */encrypt* and */decrypt*: Provide direct data encryption and decryption capabilities using a Key Encryption Key (KEK) for symmetric cryptography.

For more information, refer to the SDK documentation.

NOTE
The use of the solution in any way other than as described in the documentation is not managed. Alternatively, get in touch with Stormshield Support for clarification.

13.1 Understanding the requirements

- You must generate your KEKs and add them to the *keks.json* file:
  - For symmetric encryption/decryption, add the KEKs in the *keks* or *encrypted_keks* sections of the file,
  - For asymmetric encryption/decryption, add the KEKs in the *keys* section of the file.

  For more information, see Configuring KEKs.
- The Key Access Service module is compatible with a KMS, but the */rewrap* route is not available and will return a 404 error in KMS mode.
- For information on global requirements and recommendations, refer to section Understanding the global requirements

13.2 Configuring the Key Access Service

The Key Access Service is configured in the *kas* section of the *config.json* file. You can configure it independently for each tenant.

The template for the Key Access Service configuration block is as follows:

```
"kas": {
   "enable": false,
   "authentication": [
     {
       "discovery_uri": "_KAS_AUTHENTICATION_DISCOVERY_URI_",
       "client_id": "_KAS_AUTHENTICATION_ISSUER_"
     },
     {
       "name": "_KAS_AUTHENTICATION_API_KEY_NAME",
       "api_key": "_KAS_AUTHENTICATION_API_KEY_VALUE"
     }
   ],
```

```
        "policy_enforcement": {
          "enable": false,
          "type": "_POLICY_ENFORCEMENT_TYPE_",
          "opa_server": {
            "url": "_URL_",
            "authentication": {
              "type": "basic",
              "user_id": "_USER_ID_",
              "password": "_PASSWORD_"
            }
          }
        }
      },
```

	Description	Type	Optional/ mandatory
enable	Enables or disables the KAS module for the tenant.	Boolean	Mandatory to use the KAS module

Authentication parameter

JSON object containing the configuration that allows authenticating to the Key Access Service. There are two types of authentication: with OpenID and JWT tokens or API keys.

When using the first method, you must have configured the Identity Provider (IDP) to deliver JWT tokens with at least these fields: "iss", "aud", "exp", "iat". For more information, see Configuring the identity provider.

Stormshield recommends using OpenID authentication whenever possible, as it provides stronger security guarantees and enables advanced use cases such as ABAC (Attribute-Based Access Control).

Parameter	Description	Type	Optional/ mandatory
discovery_uri	URL to the OpenID JSON configuration file for OpenID authentication.	String	Mandatory to use OpenID authentication.
client_id	Recipient of the JWT authentication token (see RFC 7519). An entry must be added for each identity provider.	String	Mandatory to use OpenID authentication.
name	Name of the API key. The character ":" is not allowed. See RFC 2617.	String	Mandatory to use API Key authentication.
api_key	Value of the API key. Must be a valid API key provided by Stormshield.	String	Mandatory to use API Key authentication.

policy_enforcement parameter

JSON object containing the configuration of the optional OPA enforcement feature for Key Access Service. For more information, see Implementing the authorization rules with Open

Policy Agent.

Parameter			
enable	Enable the use of OPA rules for the module.	Boolean	Mandatory for each module enabled except the PKI.
type	Kind of OPA policy to use. The possible values are: • opa_local: this mode uses local files *policy.wasm* and *policy.data.json*. The file names must be adapted according to the module used. • opa_server: this mode uses a remote OPA server.	String	Mandatory if policy_enforcement.enable is set to true
opa_server: JSON object describing the parameters required to access the OPA policy server. Stormshield guarantees compatibility with OPA version 1.2.0.			
url	URL of data API exposed endpoints. For more information, see OPA documentation. *Example*: If your rego package is *stormshield.kmaas* and you have the *allow* variable in this package, your url will be: *https://opa-server/v1/data/stormshield/kmaas/allow* The authorized protocols are http and https. Stormshield strongly recommends https in production.	String	Mandatory if policy_enforcement.type is set to opa_server
authentication	JSON object describing the parameters required to authenticate to the OPA policy server. It includes the following fields: • type: Type of authentication used to connect to the policy server. The prescribed value is "basic". • user_id: Identifier of the user account used to connect to the policy server. Mandatory if authentication.type is set on "basic" • password: Password of the user account used to connect to the policy server. Mandatory if authentication.type is set on "basic"	Object	Mandatory if type is set to opa_server

EXAMPLE
The content below is provided as an example and must not be used as such in the Stormshield KMaaS KAS configuration of one of your tenants.

```
"kas": {
   "enable": true,
   "authentication": [
```

```
        {
           "discovery_uri": "https://localhost:4000/static/wrap-private-key/.well-
known/openid-configuration",
           "client_id": "cse-wrapprivatekey"
        }
     ],
     "policy_enforcement": {
        "enable": true,
        "type": "opa_server",
        "opa_server": {
           "url": "http://localhost:8181/v1/data/stormshield/kas/allow",
           "authentication": {
              "type": "basic",
              "user_id": "admin",
              "password": "admin"
           }
        }
     }
  }
```

13.3 Customizing the authorization rules

You can customize the rules that allow or deny a request to the Stormshield KMaaS, using Open Policy Agent (OPA). The policy evaluates the request inputs. If the request is forbidden, the access is denied and the "403 Forbidden" error is returned.

In the *config.json* file, the *policy_enforcement.enable* parameter is mandatory. It allows you to specify whether you want to enable OPA rules or not.

The inputs relating to all API routes are described in section Inputs relating to all API routes.

The tables below describe the inputs specific to the Key Access Service.

13.3.1 Inputs specific to the Key Access Service rewrap, encrypt and decrypt routes

As a Stormshield KMaaS administrator, you can use these inputs to create custom rules to access the Key Access Service, for example to filter the application of policies. Use some of these inputs in the created policy.

Input	Description	Source of the input
endpoint	API routes called: "*kas/rewrap*", "*kas/encrypt*", "*kas/decrypt*"	URL of the request
tenantId	unique identifier of a tenant in UUID format. Example: 2363615f-5b08-4119-a5bd-fad3f5f3f420	URL of the request
authentication.authType	Type of authentication used for the request. Possible values : • token, • apiKey	"authorization" header used.
authentication.keyName	Name of the API key used to authenticate the request. Only present when using "apiKey" authentication.	Name corresponding to the key in configuration.

Input	Description	Source of the input
authentication.iss	Entity which has created and signed the token. Only present when using "token" authentication.	JWT authentication token provided by the IDP.
authentication.aud	Corresponds to the audience for which the token was issued. Only present when using "token" authentication. Example: ['cse-authorization', 'cse-authorization1']	JWT authentication token provided by the IDP.
authentication.iat	Date when the token was issued. In timestamp format (integer) Only present when using "token" authentication. Example: 1677679386	JWT authentication token provided by the IDP.
authentication.exp	Date when the token expires. In timestamp format (integer) Only present when using "token" authentication. Example: 1677679386	JWT authentication token provided by the IDP.
authentication.customClaims	Optional. Custom claims provided par the IDP. Only present when using "token" authentication. See Inputs specific to the Key Access Service rewrap, encrypt and decrypt routes.	JWT authentication token provided by the IDP.

Inputs specific to the Key Access Service rewrap and decrypt routes

Input	Description	Source of the input
kekId	Identifier of the KEK used for data decryption.	Body of the decrypt request, or ID of the active KEK if not specified.
policy	Content of the optional policy block included in the decrypt request body. The block is optional for the *kas/rewrap* route, but mandatory for the *kas/decrypt* route You can use it to define ABAC policies using *policy.body.dataAttributes*. It is present only when the policy block is included in the */kas/rewrap* or *kas/decrypt* requests or when using Stormshield SDK.	Body of the rewrap or decrypt request.

14. Administration (Admin)

The Administration module (Admin) allows external administrators to manage keys per tenant and per module in the Stormshield KMaaS database via an authenticated REST API. It allows to secure database server access during key management operations.
In this alpha version, the Admin module only allows to get and create keys.

As the authentication is mandatory, the Stormshield KMaaS first checks the related tenant configuration. It can be either:

- User authentication with a JWT token,
- API_key authentication with a previously declared key.

The Admin module generates logs for all the operations that it performs.

For more information on API routes, refer to the API documentation.

14.1 Understanding the requirements

- The Admin module is only compatible with database key storage. In the *config.json* file, the *persistence_type* parameter must be set to "database".
- For information on global requirements and recommendations, refer to section Understanding the global requirements

14.2 Configuring the Admin module

The Admin module is configured in the *admin* section of the *config.json* file. You can configure it independently for each tenant.

The template for the Admin configuration block is as follows:

```
"admin": {

"enable": _IS_ADMIN_ENABLE_,
"authentication": [
   {
     "discovery_uri": "_ADMIN_AUTHENTICATION_DISCOVERY_URI_",
     "client_id": "_ADMIN_AUTHENTICATION_ISSUER_"
   },
   {
     "name": "_ADMIN_AUTHENTICATION_API_KEY_NAME",
     "api_key": "_ADMIN_AUTHENTICATION_API_KEY_VALUE"
   }
"policy_enforcement": {
    "enable": false,
    "type": "_POLICY_ENFORCEMENT_TYPE_",
    "opa_server": {
      "url": "_URL_",
      "authentication": {
        "type": "basic",
        "user_id": "_USER_ID_",
        "password": "_PASSWORD_"
      }
    }
  }
```

	Description	Type	Optional/ mandatory
enable	Enables or disables the Admin module for the tenant.	Boolean	Mandatory to use the Admin module

Authentication parameter

In this beta version, this parameter is mandatory but not yet used.

JSON object containing the configuration that allows authenticating to the Admin module. There are two types of authentication: with OpenID and JWT tokens or API keys.

When using the first method, you must have configured the Identity Provider (IDP) to deliver JWT tokens with at least these fields: "iss", "aud", "exp", "iat". For more information, see Configuring the identity provider.

Parameter	Description	Type	Optional/ mandatory
discovery_uri	URL to the OpenID JSON configuration file for OpenID authentication.	String	Mandatory to use OpenID authentication.
client_id	Recipient of the JWT authentication token (see RFC 7519). An entry must be added for each identity provider.	String	Mandatory to use OpenID authentication.
name	Name of the API key. The character ":" is not allowed. See RFC 2617.	String	Mandatory to use API Key authentication.
api_key	Value of the API key. Must be a valid API key provided by Stormshield.	String	Mandatory to use API Key authentication.

policy_enforcement parameter

In this beta version, this parameter is mandatory but not yet used.

JSON object containing the configuration of the optional OPA enforcement feature for Key Access Service. For more information, see Implementing the authorization rules with Open Policy Agent.

Parameter			
enable	Enable the use of OPA rules for the module.	Boolean	Mandatory for each module enabled except the PKI.

Parameter			
type	Kind of OPA policy to use. The possible values are: • opa_local: this mode uses local files *policy.wasm* and *policy.data.json*. The file names must be adapted according to the module used. • opa_server: this mode uses a remote OPA server.	String	Mandatory if policy_enforcement.enable is set to true
opa_server: JSON object describing the parameters required to access the OPA policy server. Stormshield guarantees compatibility with OPA version 1.2.0.			
url	URL of data API exposed endpoints. For more information, see OPA documentation. *Example*: If your rego package is *stormshield.kmaas* and you have the *allow* variable in this package, your url will be: *https://opa-server/v1/data/stormshield/kmaas/allow* The authorized protocols are http and https. Stormshield strongly recommends https in production.	String	Mandatory if policy_enforcement.type is set to opa_server
authentication	JSON object describing the parameters required to authenticate to the OPA policy server. It includes the following fields: • type: Type of authentication used to connect to the policy server. The prescribed value is "basic". • user_id: Identifier of the user account used to connect to the policy server. Mandatory if authentication.type is set on "basic" • password: Password of the user account used to connect to the policy server. Mandatory if authentication.type is set on "basic"	Object	Mandatory if type is set to opa_server

EXAMPLE
The content below is provided as an example and must not be used as such in the Stormshield KMaaS Admin configuration of one of your tenants.

```
"admin": {
   "enable": true,
   "authentication": [
      {
         "discovery_uri": "https://localhost:4000/static/wrap-private-key/.well-
known/openid-configuration",
         "client_id": "cse-wrapprivatekey"
      }
   ],
```

```
    "policy_enforcement": {
       "enable": true,
       "type": "opa_server",
       "opa_server": {
          "url": "http://localhost:8181/v1/data/stormshield/admin/allow",
          "authentication": {
             "type": "basic",
             "user_id": "admin",
             "password": "admin"
          }
       }
    }
}
```

14.3 Customizing the authorization rules

You can customize the rules that allow or deny a request to the Stormshield KMaaS, using Open Policy Agent (OPA). The policy evaluates the request inputs. If the request is forbidden, the access is denied and the "403 Forbidden" error is returned.

In the *config.json* file, the *policy_enforcement.enable* parameter is mandatory. It allows you to specify whether you want to enable OPA rules or not.

The inputs relating to all API routes are described in section Inputs relating to all API routes.

The tables below describe the inputs specific to the Admin module.

14.3.1 Inputs specific to all Admin routes

You can use the following inputs to create custom rules to access the Admin module, for example to filter the application of policies.

Input	Description	Source of the input
operation	API routes called: "*create_key*", "*update_key*", "*get_keys*", "*update_key*" This input replaces the *endpoint* input for the Admin routes.	URL of the request
tenantId	Unique identifier of a tenant in UUID format. Example: 2363615f-5b08-4119-a5bd-fad3f5f3f420	URL of the request
administratedModule	Name of the module. Possible values: • Kacls, • CryptoApi, • Dke, • Kas	URL of the request
authentication.authType	Type of authentication used for the request. Possible values: • token, • apiKey	"authorization" header used.

Input	Description	Source of the input
authentication.keyName	Name of the API key used to authenticate the request. Only present when using "apiKey" authentication.	Name corresponding to the key in configuration
authentication.iss	Entity which has created and signed the token. Only present when using "token" authentication.	JWT authentication token provided by the IDP.
authentication.aud	Corresponds to the audience for which the token was issued. Only present when using "token" authentication. Example: ['admin-authorization', 'admin-authorization1']	JWT authentication token provided by the IDP.
authentication.iat	Date when the token was issued. In timestamp format (integer) Only present when using "token" authentication. Example: 1677679386	JWT authentication token provided by the IDP.
authentication.exp	Date when the token expires. In timestamp format (integer) Only present when using "token" authentication. Example: 1677679386	JWT authentication token provided by the IDP.
authentication.customClaims	Optional. Custom claims provided par the IDP. Only present when using "token" authentication.	JWT authentication token provided by the IDP.

14.3.2 Inputs specific to the Admin get_key route

Input	Description	Source of the input
keyId	Identifier of the requested key.	URL of the request.

14.3.3 Inputs specific to the Admin create_key route

Input	Description	Source of the input
displayName	Name of the key to create.	Body of the request.
algorithm	Algorithm used to create the key. Possible values: • RSA-OAEP, • AES-GCM	Body of the request.

Input	Description	Source of the input
algorithmParameters.modulusLength	Length of the asymmetric key algorithm. Only present if algorithm is "RSA-OAEP". Possible values: • 2048, • 4096	Body of the request.
algorithmParameters.hash	Hash of the asymmetric key. Only present if algorithm is "RSA-OAEP". Prescribed value: • SHA-256	Body of the request.
algorithmParameters.length	Length of the symmetric key algorithm. Only present if algorithm is "AES-GCM". Prescribed value: • 256	Body of the request.

14.3.4 Inputs specific to the Admin update_key route

Input	Description	Source of the input
keyId	Identifier of the key to update.	URL of the request.
displayName	Optional. Updated name of the key.	URL of the request.
status	Optional. Updated status. Prescribed value: • compromised	URL of the request.

14.4 Using the Admin module

14.4.1 Creating keys

Two types of keys can be created:

- Symmetric keys use the AES-GCM algorithm. The following table shows the supported usage and modules:

Module	wrap	unwrap	encrypt	decrypt
KACLS	X	X		
Key Access Service	X	X		
Crypto API	X	X	X	X

Whenever a new symmetric key is created, it becomes the default key with the active status for the related tenant and module.

- Asymmetric keys use the RSA-OAEP algorithm. The following table shows the supported usage and modules:

Module	wrap	unwrap	encrypt	decrypt
Key Access Service	X	X	X	X
DKE	X	X	X	X

To create keys, use the following API route:

POST /api/v1/{tenantid}/admin/{moduleName}/key

For more information, see the API documentation.

14.4.2 Getting information about keys

Getting information about keys is only available for the modules supporting keys: KACLS, Key Access Service, DKE, Crypto API.

To get a list of keys for a module, use the following API route:

GET /api/v1/{tenantid}/admin/{moduleName}/keys

To get information about a specific key, use the following API route and add the key identifier:

GET /api/v1/{tenantId]/admin/{moduleName]/keys/{keyId}

For more information, see the API documentation.

14.4.3 Updating keys

Updating keys consists in modifying the *display_name* and *status*. This operation is only available for the modules supporting keys: KACLS, Key Access Service, DKE, Crypto API.

To update a key for a specific module, use the following API route:

PATCH /api/v1/{tenantId]/admin/{moduleName]/keys/{keyId}

If a key is compromised, it can no longer be used for encryption and decryption. Such operations on a compromised key will raise a 403 error.

For more information, see the API documentation.

15. Public Key Infrastructure (PKI)

The PKI consists of:

- A Certificate Authority (CA), which delivers certificates,
- A Registration Authority (RA), which acts as an intermediary between the CA and the end-users by handling the verification of identity and certificate signature. PKI uses an automated RA complying to the EST protocol. For more information, refer to the RFC7030 Documentation.

The PKI workflow is as follows:

1. Through an EST API endpoint, *.well-known/est/simpleenroll*, the users send PKCS#10 Certificate Signing Requests (CSR) to the default PKI engine. A PKI engine is the combination of a specific CA and RA.
2. The RA engine verifies the CSR information.
3. The CA issues the certificates.

Several PKI engines can be defined for the same tenant in the Stormshield KMaaS configuration file. However, only a single one, called the default PKI engine, can be used at a time

WARNING
The PKI is restricted to RSA and ECC algorithms and to the certificate extensions required for mTLS. For more information refer to section Compatibility of algorithms and CA properties.

NOTE
The use of the solution in any way other than as described in the documentation is not managed. Alternatively, get in touch with Stormshield Support for clarification.

15.1 Understanding the requirements

- For information on global requirements and recommendations, refer to section Understanding the global requirements

- You must have at least one Certification Authority with the following PEM files. Both files must be stored in the */etc/stormshield/pki/* directory. You can name them as you wish: the file names in the table are examples.
  These files will be referenced in the *config.json* file. For more information, see Certification authority (CA) parameters

Description	File name example	Parameter in config.json
File containing the CA certificate (and its trust chain) which will issue the users' certificates.	*ca_certificate.pem*	full_chain_certificates_path
File containing the CA private key used to sign the certificate issued. Do not set a password, as private keys with password are not supported. No specific key usage is required.	*ca_private_key.pem*	key_path

  For an example on how to create these files, see Creating a CA with OpenSSL.
  For information on supported algorithms, extensions, key usages etc, see Compatibility of algorithms and CA properties.

15.2 Compatibility of algorithms and CA properties

15.2.1 Algorithms

The current version of the PKI supports the following algorithms:

- *RSASSA-PKCS v1.5* and *ECDSA with Prime256-V1* for the Certificate Signing Request (CSR),
- *RSASSA-PKCS v1.5* for signing the issued certificate with the CA private key. This is the OpenSSL standard RSA algorithm.

Algorithms	CSR	CA
RSA (RSASSA-PKCS-v1.5)	Supported	Supported
EC (ECDSA with Prime256-V1)	Supported	Not supported

  Stormshield recommends using 4096-bit RSA keys (at least 2048 bits) to be as secure as possible in this context.
- *RsaWithSha256, RsaWithSha384* and *RsaWithSha512* for the hash.
  It is specified in the *ca.algorithms.hash* configuration parameter of the *config.json* file.

For more information, refer to Configuring the PKI.

15.2.2 CA properties

The certification authority of PKI does not support all attributes and extensions. Unsupported ones are ignored. The following sections list the supported properties and their expected behaviors.

Common properties

The second column of the tables contains the Objet Identifier (OID) corresponding to the property.

A Distinguished Name (DN) is the full identity of a certificate or CSR, and is composed of multiple attributes such as Country (C), Locality (L), Common Name (CN), etc.

The following **Distinguished Name attributes** are supported:

DN attribute	OID
common name	2.5.4.3
surname	2.5.4.4
serial name	2.5.4.5
country name	2.5.4.6
locality name	2.5.4.7
state or province name	2.5.4.8
street address	2.5.4.9
organization name	2.5.4.10
organizational unit name	2.5.4.11
title	2.5.4.12
business category	2.5.4.15
postal code	2.5.4.17
telephone number	2.5.4.20
name	2.5.4.41
given name	2.5.4.42
initials	2.5.4.43
generation qualifier	2.5.4.44
unique identifier	2.5.4.45
distinguished name qualifier	2.5.4.46
pseudonym	2.5.4.65
email address	1.2.840.113549.1.9.1
user id	0.9.2342.19200300.100.1.1
domain component	0.9.2342.19200300.100.1.25
DN attribute	OID
common name	2.5.4.3
surname	2.5.4.4
serial name	2.5.4.5
country name	2.5.4.6
locality name	2.5.4.7
state or province name	2.5.4.8

DN attribute	OID
street address	2.5.4.9
organization name	2.5.4.10
organizational unit name	2.5.4.11
title	2.5.4.12
business category	2.5.4.15
postal code	2.5.4.17
telephone number	2.5.4.20
name	2.5.4.41
given name	2.5.4.42
initials	2.5.4.43
generation qualifier	2.5.4.44
unique identifier	2.5.4.45
distinguished name qualifier	2.5.4.46
pseudonym	2.5.4.65
email address	1.2.840.113549.1.9.1
user id	0.9.2342.19200300.100.1.1
domain component	0.9.2342.19200300.100.1.25

The following x509 v3 extensions are supported.

Extension	OID
basic constraints	2.5.29.19
key usage	2.5.29.15
extended key usage	2.5.29.37
subject key identifier	2.5.29.14
subject alternative name	2.5.29.17
authority key identifier	2.5.29.35

IMPORTANT
The CA of the PKI does not add its own extensions to the CSR during the certificate issuance process. It only keeps the CSR supported extensions. It is the administrator's responsability to correctly set the required extensions in the CSR OpenSSL configuration.

The tables below indicate whether an extension sub-category is supported:

Basic constraints	Supported/Not supported
CA: boolean	Supported. Mandatory with value="true" for root certificates.
pathlen: number	Supported.
critical:string	Supported. Mandatory for root certificates.

Key usage	Supported/Not supported
decipherOnly	Supported
encipherOnly	Supported
digitalSignature	Supported
nonRepudiation	Supported
keyEncipherment	Supported
dataEncipherment	Supported
keyAgreement	Supported
keyCertSign	Supported
cRLSign	Supported

Extended key usage	Supported/Not supported
serverAuth	Supported
clientAuth	Supported
codeSigning	Not supported
emailProtection	Not supported
timeStamping	Not supported
OCSPSigning	Not supported
ipsecIKE	Not supported
msCodeInd	Not supported
msCodeCom	Not supported
msCTLSign	Not supported
msEFS	Not supported

15.2.3 Other CSR-specific properties

Only mTLS-specific subject alternative names are supported. The others are ignored.

Subject alternative names	Supported/Not supported
DNS	Supported
IP	Supported
email	Not supported
RID	Not supported
dirName	Not supported
otherName	Not supported
Subject alternative names	Supported/Not supported
DNS	Supported
IP	Supported
email	Not supported
RID	Not supported
dirName	Not supported
otherName	Not supported

15.3 Configuring the PKI

The PKI is configured in the *pki* section of the *config.json* file. You can configure it independently for each tenant.

The template for the PKI configuration block is as follows:

```
"pki": {
        "enable": false,
        "default": "_PKI_DEFAULT_PKI_",
        "pki_engines": [
          {
            "id": "_PKI_ENGINE_ID_",
            "name": "_PKI_ENGINE_NAME_",
            "ra": {
              "type": "_PKI_ENGINE_RA_TYPE_",
              "authentication": [
                {
                  "name": "_PKI_RA_API_KEY_NAME",
                  "api_key": "_PKI_RA_API_KEY_VALUE"
                }
              ]
            },
            "ca": {
              "type": "_PKI_ENGINE_CA_TYPE_",
              "key_path": "_PKI_ENGINE_KEY_PATH_",
              "full_chain_certificates_path": "_PKI_ENGINE_CERT_CHAIN_PATH_",
              "certificate_signature": {
                "algorithms": {
                  "hash": "_PKI_ENGINE_HASH_ALGO_",
                  "signature": "_PKI_ENGINE_SIGNATURE_ALGO_"
                },
                "parameters": {
```

```
                "salt_length": 20
              }
            },
            "certificate_profiles": {
              "validity_period": 31536000
            }
          }
        }
      ]
    },
```

	Description	Type	Optional/ mandatory
enable	Enables or disables the PKI for the tenant.	Boolean	Mandatory to use the PKI module
default	Identifier of the tenant default PKI engine. It must match one of the *id* parameter of the pki_ engines object. It is used as the default PKI engine configuration.	String	Mandatory
pki_engines: JSON object array containing the list of all PKI engines as JSON objects.			
id	Unique identifier of the PKI engine in UUIDv4 format.	String	Mandatory
name	Name of the PKI engine.	String	Mandatory
ra: JSON object containing the registration authority configuration. For more details, see the table below.			
ca: JSON object containing the certification authority configuration. For more details, see the table below.			

Registration authority (RA) parameters

Parameter	Description	Type	Optional/ mandatory
type	Type of registration autority. The only prescribed value is "est".	String	Mandatory
authentication: JSON object containing the list of API keys.			
name	Name of the API key. The character ":" is not allowed. See RFC 2617.	String	Mandatory
api_key	Value of the API key.	String	Mandatory

Certification authority (CA) parameters

Parameter	Description	Type	Optional/ mandatory
type	Type of the certification authority. The only prescribed value is "local".	String	Mandatory

Parameter	Description	Type	Optional/ mandatory
key_path	Path to the CA private key in PEM format. See Configuring the PKI.	String	Mandatory
full_chain_certificates_path	Path to the CA certificate chain in PEM format. See Configuring the PKI.	String	Mandatory
certificate_signature: JSON object containing the metadata for the signature of the certificate signing request (CSR) by the CA.			
algorithms	Signature algorithm. It includes the following fields: • signature: The only prescribed value is "rsassa_pkcs1_v1_5", • hash: The prescribed values are "sha-256" or "sha-384", and "sha-512". See Algorithms.	String	Mandatory
parameters	Metadata for specific algorithms. It includes the following field: • salt_length: RSA-PSS specific signature parameter to improve security (Not used).	Integer	Optional
certificate_profiles: Metadata of the certificate profile.			
validity_period	Validity period of issued certificate in seconds. The minimum is 1, and the maximum is 2 147 483 647 seconds (68 years). A typical value is 31 536 000 seconds (1 year).	Integer	Mandatory

EXAMPLE
The content below is provided as an example and must not be used as such in the Stormshield KMaaS PKI configuration of one of your tenants.

```
"pki": {
  "enable": true,
  "default": "8e071476-01a2-44e8-90f3-be94d2de46ef",
  "pki_engines": [
    {
      "id": "8e071476-01a2-44e8-90f3-be94d2de46ef",
      "name": "pki_name",
      "ra": {
        "type": "est",
        "authentication": [
          {
            "name": "testApiKey",
            "api_key": "VW0FmFl73leUkYGBCr8DjlFcZBt6en5p"
          }
        ]
      },
```

```
        "ca": {
          "type": "local",
          "key_path": "/etc/stormshield/pki/ca_private_key.pem",
          "full_chain_certificates_path": "/etc/stormshield/pki/ca_certificate.pem",
          "certificate_signature": {
            "algorithms": {
              "hash": "sha-256",
              "signature": "rsassa_pkcs1_v1_5"
            },
            "parameters": {
            }
          },
          "certificate_profiles": {
            "validity_period": 3153600
          }
        }
      }
    ]
  }
```

15.4 Issuing certificates

Before certificates can be issued, you must:

- Set up the CA configuration for the tenant. See Configuring the PKI.
- Make sure that the CA chain is trusted.

15.4.1 Issuing a standard certificate

1. Create a CSR. The only mandatory attribute is the Common Name.
   For an example of CSR creation with OpenSSL, see Issuing a mTLS certificate with a CSR.
2. POST the CSR content to the *URI/{tenantId}/.well-known/est/simpleenroll* endpoint with the following headers:
   - Content-Type must be `application/pkcs10`,
   - Content-Transfer-Encoding must be `base64`.

   > **NOTE**
   > The PKI supports CSR content with or without PEM PKCS#10 headers/footers, and with or without line breaks.

   If the operation is successful, the response, `Content-Type : application/pkcs7-mime`, is a base64 string in PEM style (line breaks every 76 characters), containing the issued certificate in PKCS#7.
   If it fails, the response if `Content-Type : application/json` and contains an error JSON object.

   > **NOTE**
   > PKCS#7 is a widely supported cryptographic message syntax mandated by the EST protocol. It allows to return in a compact way multiple certificates, e.g., the whole certificate chain, in addition to the newly issued certificate, and also other metadata if necessary.

For more information on the API routes, see the API Documentation.

15.4.2 Issuing a certificate with common name override

The PKI allows generating a CSR before the final common name is known.

If the Common Name specified in the CSR is not suitable, you can override it by adding the *x-override-cn* header in the */simpleenroll* API call. The certificate issued will contain the Common Name specified in the header.

The Common Name in the *x-override-cn* header must:

- be a non-empty string when the header is defined,
- have less than 64 characters,
- contain only these characters:
  `ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_.~!#$'`'`"()*+,/:;=?@[] %`
  and/or the following percent-encoding characters:
  `%20%21%22%23%24%25%27%28%29%2A%2B%2C%2F%3A%3B%3D%3F%40%5B%5D`

For more information on the API routes, see the API Documentation.

15.5 Testing use cases with OpenSSL

15.5.1 Creating a CA with OpenSSL

1. Create the *ca_certificate_config.cnf* OpenSSL configuration file for the certification authority (CA):

```
[ req ]
default_bits       = 4096
default_keyfile    = ca_private_key.pem
distinguished_name = req_distinguished_name
x509_extensions    = req_ext
prompt             = no

[ req_distinguished_name ]
CN = your CA Name
C  = your country
L  = your locality
O  = your organization

[ req_ext ]
basicConstraints = critical, CA:true
keyUsage = critical, digitalSignature, cRLSign, keyCertSign
```

2. Generate the RSA private key:
   `# openssl genpkey -des3 -algorithm RSA -out ca_private_key.pem \-pkeyopt rsa_keygen_bits:4096`
3. Extract the public key:
   `# openssl rsa -in ca_private_key.pem -pubout -out ca_public_key.pem`
4. Generate an autosigned root certificate (10-year validity period):
   `# openssl req -x509 -new -key ca_private_key.pem -out ca_certificate.pem -days 3650 \-config ca_certificate_config.cnf`
5. Check the root certificate:
   `# openssl x509 -in ca_certificate.pem -text -noout`
6. Move the certificate file (ca_certificate.pem) and the private key (ca_private_key.pem) to the */etc/stormshield/pki* directory.

15.5.2 Issuing a mTLS certificate with a CSR

1. Generate the related *mtls_client_csr.cnf* CSR configuration file:

```
[ req ]
distinguished_name = req_distinguished_name
req_extensions     = req_ext
string_mask        = utf8only
prompt             = no

[ req_distinguished_name ]
C  = your country
L  = your locality
O  = your organization
CN = your common name

[ req_ext ]
keyUsage = critical, digitalSignature, keyEncipherment,
keyAgreementextendedKeyUsage = serverAuth, clientAuth
basicConstraints = critical
```

2. Generate the CSR private key:
   `# openssl genpkey -algorithm RSA -out mtls_client_private_key.pem \ -pkeyopt rsa_keygen_bits:4096`
   The certification authority (CA) private keys are extremely sensitive items in terms of security. You must follow the ANSSI recommendations concerning their life cycles.
3. Generate the CSR:
   `# openssl req -new -key mtls_client_private_key.pem \-out mtls_ client.csr -config mtls_client_csr.cnf -extensions req_ext`
4. Check the CSR:
   `# openssl req -text -noout -in mtls_client.csr`
5. POST the *mtls_client.csr* content to the *URI/{tenantId}/.well-known/est/simpleenroll* endpoint with the following headers:
   - Content-Type must be `application/pkcs10`,
   - Content-Transfer-Encoding must be `base64`.

   The response, `Content-Type : application/pkcs7-mime`, is a base64 string in PEM style (line breaks every 76 characters), containing the issued certificate in PKCS#7.
6. If the PKCS#7 content was saved in a *mtls_client_certificate.p7* file, extract the issued certificate using these OpenSSL commands:
   `# openssl base64 -d -in mtls_client_certificate.p7 | \openssl pkcs7 -inform DER -outform PEM -print_certs \-out mtls_client_ certificate.pem`
7. Use both *mtls_client_private_key.pem* and *mtls_client_certificate.pem* in the HTTPS agent of your service or with CURL to activate mTLS authentication.

16. Implementing the authorization rules with Open Policy Agent

You can customize the rules that allow or deny a request to the Stormshield KMaaS, using Open Policy Agent (OPA). The policy evaluates the request inputs. If the request is forbidden, the access is denied and the "403 Forbidden" error is returned.

EXAMPLE
You can define a policy allowing access to the Stormshield KMaaS only to users from the *stormshield.eu* domain.

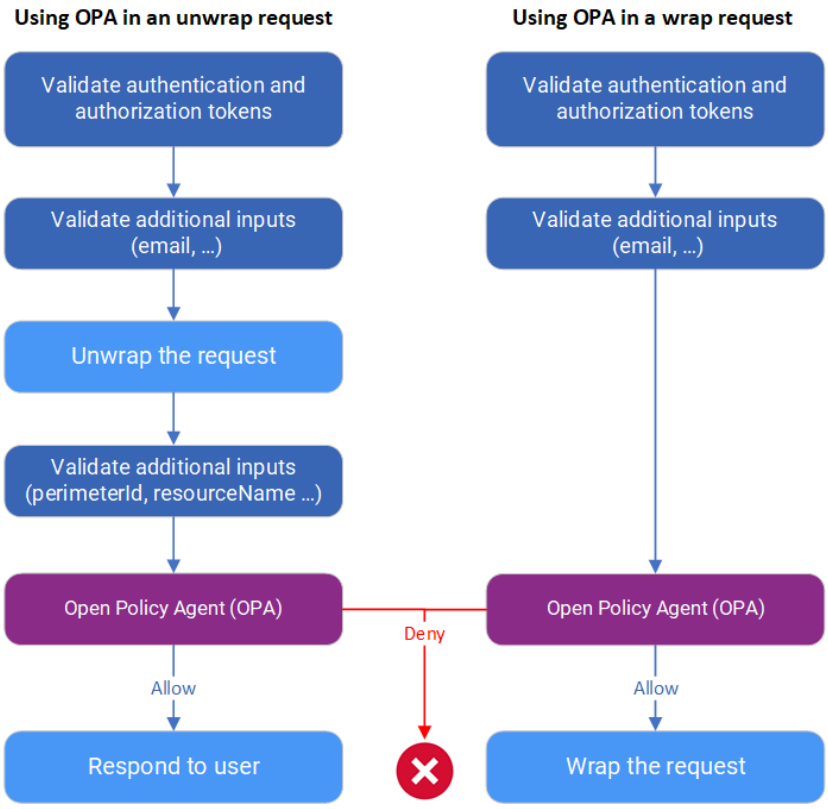
You can add an OPA policy to the following API routes. If you specify rules for other routes, they will be ignored.

- wrap, unwrap, privilegedwrap, privilegedunwrap, rewrap, certs, digest, wrapprivatekey, privatekeydecrypt, privilegedprivatekeydecrypt,privatekeysign, encrypt, decrypt, and rewrap.

The policy enforcement is configured for each tenant and module(i.e., KACLS, Crypto API, KAS). For more information, refer to sections:

- Configuring the KACLS
  Customizing the authorization rules
- Configuring Crypto API
  Customizing the authorization rules
- Configuring the Key Access Service
  Customizing the authorization rules

The diagram below indicates at which stage of the requests the OPA policy is applied for the "wrap" and "unwrap" requests.



16.1 Defining an OPA policy

There are two different OPA modes, local and server.

16.1.1 Local OPA mode

With the local OPA mode, the users use the local files located in the */etc/stormshield/cse* directory.

If using Stormshield KMaaS on several instances, you must apply the following procedure on all instances.

1. Edit the two files needed to define a policy for the module you are implementing: the KACLS, Crypto API, or Key Access Service:

Module	File name	Description
KACLS	policy.wasm	• This file defines the policy rules, based on request inputs • It is generated from a rego file, • Use the OPA command line tool to generate it, • Use The rego Playground to develop and test a policy.
Crypto API	policy-crypto-api.wasm	
KAS	policy-kas.wasm	
DKE	policy-dke.wasm	
Admin	policy-admin.wasm	
KACLS	policy.data.json	• This file contains data that can be referenced in the rego file. • It makes it possible to add variables to the *policy.wasm* file so that you do not to have to recompile the file each time you modify it.
Crypto API	policy-crypto-api.data.json	
KAS	policy-kas.data.json	
DKE	policy-dke.data.json	
Admin	policy-admin.data.json	

In RPM mode, both these files already exist in */etc/stormshield/cse* only for the KACLS. They define a default policy which allows all requests. If you want to create your own policy, customize these files using the inputs and the examples provided in the following sections.
In Docker mode, these default policy files are provided along with the Docker image. You can place them in the folder containing your configuration files, or define your own.

The *.wasm* and *.json* files are mandatory if the *policy_enforcement.enable* parameter value is set to true and if the type is *opa_local* in the *config.json* file. For more information, see Configuring the KACLS Configuring Crypto API, or Configuring the Key Access Service.
If one of the files is not present and the parameter value is set to true, the service will not start and the corresponding module (i.e., KACLS, Crypto API or KAS) will be disabled. A log is also issued to indicate that the policy is disabled.

2. Run the restart command to take into account the modified *.wasm* and *.json* files. If one of the files is not valid, the service does not start and a log is issued. For more information, refer to the section the *Log Guide*.

16.1.2 OPA Server

With the OPA server mode, the users contact an OPA server to get the result of an online policy. This mode allows you to centralize the policy and avoids duplicating the file on several instances in the case of a multi-instance architecture.

With this mode, you must implement the verification of Basic authentication, either via the tools provided by OPA or via another method such as a gateway.

To use this mode in your configuration, in the *config.json* file, section *policy_enforcement*, set the *type* parameter to *opa_server* for the KACLS, Crypto API, or the Key Access Service.

For more information, refer to sections Configuring the KACLS, Configuring Crypto API or Configuring the Key Access Service.

To configure the policies on your OPA server, refer to the OPA Rest API documentation . Stormshield guarantees compatibility with OPA version 1.2.0.

16.2 Inputs relating to all API routes

These inputs are meant to create customized rules to access the Stormshield KMaaS. Use them to filter the application of policies.

You can use these inputs in the custom policy.

Input	Description	Source of the input
endpoint	API routes called: "wrap", "unwrap", "privilegedwrap", "privilegedunwrap", "rewrap", "certs", "wrapprivatekey", "privilegedprivatekeydecrypt", "digest", "privatekeydecrypt", "privatekeysign", "crypto/encrypt", "crypto/decrypt", "kas/encrypt", "kas/decrypt", "kas/rewrap", "dke/:kid", and "dke/:kid/versionId/Decrypt".. For Admin routes, this property is replaced by *operation*.	URL of the request
tenantId	unique identifier of a tenant in UUID format. Example: 2363615f-5b08-4119-afdb-fad3f5f3f420.	URL of the request
module	The Stormshield KMaaS module associated to the API route. Possible values: • "Kacls" • "CryptoApi" • "Kas" • "dke"	URL of the request

16.3 Example of policy implementation

In this example, a rule is added which allows only the users present in the *authorizedUsers* list to perform a request for the 'unwrap' route used for Google Drive.

In the *policy.rego* file:

- *input* refers to the data provided by the Stormshield KMaaS to the policy.
- *data* refers to the data in the *policy.data.json* file.

Once the *policy.rego* file has been compiled into *policy .wasm* using via the OPA tool, you can add or remove authorized users by updating the content of the *authorizedUsers* field in the *policy.data.json* file.

In this example, a user with an email address in the authentication token 'user1@test.com' or 'user2@test.com' will be allowed to use the 'unwrap' route to decrypt a Google Drive document, whereas other users will not be allowed to do so. Other users will be allowed to use the "unwrap" route if it does not imply Google Drive.

The *.rego* examples provided in this documentation use a syntax that is still supported by OPA but is no longer the recommended style. These examples are not meant for writing production policies. Stormshield recommends following the official OPA documentation, which reflects the most up-to-date best practices.

16.3.1 policy.rego file

```
package cse
```

```
# ------------------------------------------------------

# Deny by default
default allow := false

# Allow all other endpoints
allow if {
input.endpoint != "unwrap"
}

# Allow access to unwrap endpoint if not concerning drive application
allow if {
input.endpoint == "unwrap"
input.authorization.iss != "gsuitecse-tokenissuer-drive@system.gserviceaccount.com"
}

# Allow access to unwrap concerning drive, only for authorizedUsers
allow if {
input.endpoint == "unwrap"
input.authorization.iss == "gsuitecse-tokenissuer-drive@system.gserviceaccount.com"
input.authentication.email in data.authorizedUsers
}
```

16.3.2 policy.data.json file

```
{
"authorizedUsers": ["user1@test.com", "user2@test.com"]
}
```

16.4 Using custom claims

Authentication and authorization tokens may contain user data (claims) that are not required by the Stormshield KMaaS.

Such data is placed in a “customClaims” object and can be used in an OPA policy file.

For example, the Stormshield KMaaS may get an authentication token of the following form:

```
{
    iss: 'issuer-authentication',
    aud: 'cse-authentication' ,
    exp: 1731599885,
    iat: 1728917885,
      email: 'user@domain.com',
      user_age: 23
 }
```

The "user_age" property is not required by the Stormshield KMaaS. However, it will be transmitted to OPA in a “customClaims” object, as follows:

```
{
    iss: 'issuer-authentication',
    aud: 'cse-authentication' ,
    exp: 1731599885,
```

```
    iat: 1728917885,
      email: 'user@domain.com',
      customClaims: {
          user_age: 23
      }
}
```

You can use the "user_age" property in the *policy.rego* file, using the *customClaims.{key}* syntax. In the example below, access is restricted to users over the age of 18:

```
package cse

# Deny by default
default allow := false

# Allow access if age is more than 18
allow if {
  input.authentication.customClaims.user_age >= 18
}
```

16.5 Using Attribute-based access control (ABAC)

Attribute-based access control (ABAC) is an authorization model that evaluates attributes, rather than roles, to determine access. For instance, with ABAC, the policy server can grant or deny decryption based on attributes contained in the authentication JWT token issued by the identity provider (e.g., user location, age, data sensitivity).

The IDP compares the following attributes:

- The custom claims delivered to the user by the IDP in the authentication JWT token,
- The attributes of the 'decrypt' route (i.e., body dataAttributes). The attributes can be either attached to the data in the Stormshield SDK or sent along with the data request using the optional "policy" property.

If the attributes do not match, access is denied and the user is not allowed to decrypt the data.

The *policy.rego* file below is an example of ABAC policy on the 'decrypt' route that compares the *location* custom claim of the authentication JWT token and the *location* attribute in the request:

```
package cse

# Deny by defaultdefault
allow := false

# Allow all routes except decrypt
allow if {
    not input.endpoint in ["decrypt"]
}

allow if {
    input.endpoint in ["decrypt"]
    some attribute in input.policy.body.dataAttributes
    attribute.location == input.authentication.customClaims.location
}
```

In this example, if:

- the 'decrypt' request contains the attribute *location:France*,
- the JWT token contains the custom claim *location:France*,

then, decryption is allowed.

However, if:

- the 'decrypt' request contains the attribute *location:France*,
- the JWT token contains the custom claim *location:Germany*

then, decryption is denied.

17. Managing logs

The Stormshield KMaaS generates logs for every operation, making it possible to trace all operations performed and potential issues. Logs are generated in JSON format. In RPM mode, the logs are managed by the *systemd* service.

There are two different log formats:

- Refer to the *Stormshield KMaaS Log Guide (v1 format)* guide for more information on logs in the old format.
- Refer to the to the *Stormshield KMaaS Log Guide* for more information on logs in the new format.

Appendix A. Creating the database schema

If you plan to use a database to store the keys, you must create a PostgreSQL 16 database schema and set the "persistence_type" parameter to "database" in the *config.json* file. For more information, see database parameter.

1. To create the schema, connect to your database server, then run the following database creation scripts:

```
SET search_path TO <_SCHEMA_>;

-- TENANT
CREATE TABLE IF NOT EXISTS tenant (
  tenant_id UUID PRIMARY KEY,
  display_name TEXT NOT NULL,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- MODULE
CREATE TABLE IF NOT EXISTS module_ref (
  module_id INTEGER PRIMARY KEY,
  display_name TEXT NOT NULL
);

INSERT INTO module_ref (module_id, display_name)
    VALUES
        (1, 'kacls'),
        (2, 'kas'),
        (3, 'crypto_api'),
        (4, 'pki'),
        (5, 'dke');

-- KEY_ALGORITHM
CREATE TABLE IF NOT EXISTS key_algorithm_ref (
    key_algorithm_id INTEGER PRIMARY KEY,
  display_name TEXT NOT NULL
);
INSERT INTO key_algorithm_ref (key_algorithm_id, display_name)
    VALUES
        (1, 'AES-GCM'),
        (2, 'RSA-OAEP');

-- KEY_STATUS
CREATE TABLE IF NOT EXISTS key_status_ref (
    key_status_id INTEGER PRIMARY KEY,
  display_name TEXT NOT NULL
);
INSERT INTO key_status_ref (key_status_id, display_name)
    VALUES
        (1, 'pre-activation'),
        (2, 'active'),
        (3, 'deactivated'),
        (4, 'compromised'),
        (5, 'suspended'),
        (6, 'destroyed');

-- KEY_LOCATION_TYPE
CREATE TABLE IF NOT EXISTS key_location_ref (
  key_location_id INTEGER PRIMARY KEY,
  display_name TEXT NOT NULL
);
INSERT INTO key_location_ref (key_location_id, display_name)
    VALUES
        (1, 'env'),
```

```
        (2, 'database');

-- MODULE_TENANT
CREATE TABLE IF NOT EXISTS MODULE (
    MODULE_ID INTEGER REFERENCES MODULE_REF (MODULE_ID),
    TENANT_ID UUID REFERENCES TENANT (TENANT_ID),
    DEFAULT_ENCRYPTION_KEY_ID UUID,
    PRIMARY KEY (MODULE_ID, TENANT_ID)
);

-- KEY
CREATE TABLE IF NOT EXISTS KEY (
    KEY_ID UUID PRIMARY KEY,
    TENANT_ID UUID NOT NULL,
    MODULE_ID INTEGER NOT NULL,
    KEY_ALGORITHM_ID INTEGER NOT NULL REFERENCES KEY_ALGORITHM_REF (KEY_
ALGORITHM_ID),
    KEY_ALGORITHM_PARAMS JSONB NOT NULL,
    DISPLAY_NAME TEXT NOT NULL,
    KEY_STATUS_ID INTEGER NOT NULL,
    KEY_USAGE INTEGER NOT NULL,
    CREATED_AT TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    UPDATED_AT TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- ADD FK CONSTRAINTS
ALTER TABLE module
    ADD CONSTRAINT module_key_fk FOREIGN KEY (default_encryption_key_id)
REFERENCES key(KEY_ID);

ALTER TABLE key
    ADD CONSTRAINT key_module_fk FOREIGN KEY (tenant_id, module_id) REFERENCES
module(tenant_id, module_id);

-- MKEK
CREATE TABLE IF NOT EXISTS mkek (
    MKEK_ID UUID PRIMARY KEY,
    TENANT_ID UUID NOT NULL REFERENCES TENANT (TENANT_ID),
    KEY_ALGORITHM_ID INTEGER NOT NULL REFERENCES KEY_ALGORITHM_REF (KEY_
ALGORITHM_ID),
    KEY_STATUS_ID INTEGER NOT NULL REFERENCES KEY_STATUS_REF (KEY_STATUS_ID),
    DISPLAY_NAME TEXT NOT NULL,
    key_location_id INTEGER NOT NULL REFERENCES KEY_LOCATION_REF (key_location_
id),
    key_value BYTEA NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- SYMMETRIC VERSION
CREATE TABLE IF NOT EXISTS symmetric_version (
    sym_key_version_id UUID PRIMARY KEY,
    key_id UUID REFERENCES key(key_id),
    encrypted_by_mkek_id UUID REFERENCES mkek(mkek_id),
    encrypted_with_crypto_context JSONB NOT NULL,
    key_location_id INTEGER NOT NULL REFERENCES KEY_LOCATION_REF (key_location_
id),
    key_value BYTEA NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- ASYMMETRIC VERSION
CREATE TABLE IF NOT EXISTS asymmetric_version (
    asym_key_version_id UUID PRIMARY KEY,
    key_id UUID REFERENCES key(key_id),
  encrypted_by_mkek_id UUID REFERENCES mkek(mkek_id),
    encrypted_with_crypto_context JSONB NOT NULL,
    key_location_id INTEGER NOT NULL REFERENCES KEY_LOCATION_REF (key_location_
id),
```

```
    private_value BYTEA NOT NULL,
    public_value BYTEA NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

2. Grant privileges to the database user:

```
CREATE USER _USER_ WITH PASSWORD _PASSWORD_;
GRANT USAGE on SCHEMA _SCHEMA_ TO _USER_;
ALTER DEFAULT PRIVILEGES IN SCHEMA _SCHEMA_ GRANT ALL PRIVILEGES ON TABLES TO _USER_;
```

18. Further reading

Additional information and answers to questions you may have about Stormshield KMaaS are available on the Documentation website and in the Stormshield knowledge base (authentication required).

documentation@stormshield.eu