



# eIDAS Middleware Documentation

*Release 4.0.1*

**Governikus GmbH & Co. KG**

**Jan 26, 2026**

## Contents

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Introduction to the German eID System</b>	<b>2</b>
<b>3</b>	<b>Introduction to Docker</b>	<b>4</b>
<b>4</b>	<b>Introduction to the VirtualBox Image</b>	<b>5</b>
4.1	First time login via console . . . . .	5
4.2	Regenerate the SSH server key . . . . .	5
4.3	Setting up network access . . . . .	5
4.4	DNS configuration . . . . .	6
4.5	Firewall configuration . . . . .	6
<b>5</b>	<b>Requirements</b>	<b>8</b>
5.1	Software Requirements . . . . .	8
5.2	Hardware Requirements . . . . .	8
5.3	Network Requirements . . . . .	8
5.4	HSM support . . . . .	9
<b>6</b>	<b>Configuration of the eIDAS Middleware application</b>	<b>10</b>
6.1	Necessary keys and certificates . . . . .	10
6.2	Setup the application.properties . . . . .	12
6.3	Startup . . . . .	14
6.4	Using the admin interface . . . . .	14
<b>7</b>	<b>Operating the server</b>	<b>17</b>
7.1	Startup and shutdown . . . . .	17
7.2	Obtain authorization certificate . . . . .	18
7.3	Additional information . . . . .	19
7.4	Monitoring . . . . .	23
7.5	Test mode . . . . .	26
<b>8</b>	<b>eIDAS Demo Application</b>	<b>28</b>
8.1	Configuration of the eIDAS Demo Application . . . . .	28
8.2	Using the eIDAS Demo Application . . . . .	29

8.3	Using the eIDAS Demo Application in Docker . . . . .	30
<b>9</b>	<b>Changelog</b>	<b>31</b>
<b>10</b>	<b>Glossary</b>	<b>39</b>
<b>Index</b>		<b>41</b>

# 1

## Overview

You have received the eIDAS Middleware in order to setup the infrastructure to support the German eID card via your eIDAS connector.

The eIDAS Middleware is distributed in three different ways:

1. Virtual Machine (OVA Format)
2. Docker Image
3. Spring Boot Application

You can decide if you want to set up your environment by yourself using just the Spring Boot JAR File or if you want to use the existing environments of the Virtual Machine or the Docker Image. The following documentation will provide examples for all three options.

The source code of the eIDAS Middleware is hosted on [GitHub](#)<sup>1</sup>.

The Docker Images are hosted on [Docker Hub](#)<sup>2</sup>.

This documentation consists of the following chapters:

- *Introduction to the German eID System* is meant to give you an overview of the German eID system.
- In the chapter *Requirements* you will find some requirements regarding hardware, software and network.
- The chapter *Configuration of the eIDAS Middleware application* will guide you through the first-time configuration of the Middleware.
- The chapter *Operating the server* will guide you on how to operate the Middleware.
- The *Changelog* lists the changes of the releases and will advise you for possible migration problems.
- At the end you will find the *Glossary*.

---

<sup>1</sup> <https://github.com/governikus/eidas-middleware>

<sup>2</sup> <https://hub.docker.com/u/governikus>

## Introduction to the German eID System

The eIDAS Middleware performs the server side of the authentication procedure with the German eID. It is an eIDAS Service providing cross-border authentication. In contrast to an eIDAS Proxy Service which is operated by the Sending MS, the eIDAS Middleware is provided by the Sending MS and every Receiving MS operates its own eIDAS Middleware instance.

The eIDAS Middleware consists of two parts: Firstly, the Middleware contains an eID server to communicate with German eID backend systems and with the user's browser and eID client. Secondly, the Middleware contains a SAML adapter to communicate with the eIDAS connector of the Member State.

It should also be mentioned that the term *service provider* is ambiguous in this specific scenario: In terms of the eIDAS environment, the service provider receives the authorization data from the eIDAS connector to authenticate the user. However, in terms of the German eID system the service provider is the SAML adapter of the Middleware, because this is the instance to which the eID server sends the data from the eID card.

The German eID scheme fulfills all requirements of the eIDAS Level of Assurance **high**. One of the major security-by-design feature is strong and mutual authentication between the relying party and the user. For example, using authorization certificates is a strong cryptographic binding between the service provider and the user. It is highly recommended to read the [German eID Whitepaper<sup>3</sup>](#) in order to understand the German eID scheme. For additional information see the page about the [eIDAS Notification of the German eID<sup>4</sup>](#) and the [eID Infrastructure<sup>5</sup>](#).

The following brief description facilitates the understanding and configuration of the eIDAS Middleware.

A single eIDAS Middleware supports multiple *eID Service Providers* and multiple eIDAS connectors. An *eID Service Provider* needs an *Authorization Certificate* to access data on the eID Card. The Authorization Certificate is issued to the *eID Service Provider* by the *Authorization CA*, also called *BerCA*. The eIDAS Middleware technically receives the *Authorization Certificate* via SOAP requests to the *Authorization CA*. These requests are secured by TLS client authentication and signed with a so-called *Request Signer Certificate* or the currently valid *Authorization Certificate*. Therefore, the public certificates of the *Authorization CA* and your eIDAS Middleware must be exchanged before the eIDAS Middleware requests the first Authorization Certificate. If you decide to use a *Request Signer Certificate* (which is highly recommended as it will become mandatory at some point), this certificate must be given to the

<sup>3</sup> [https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/EIDAS/German\\_eID\\_Whitepaper\\_v1-4.pdf?\\_\\_blob=publicationFile&v=2](https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/EIDAS/German_eID_Whitepaper_v1-4.pdf?__blob=publicationFile&v=2)

<sup>4</sup> [https://www.bsi.bund.de/EN/Themen/Oeffentliche-Verwaltung/Elektronische-Identitaeten/Online-Ausweisfunktion/eIDAS-Notifizierung/eidas-notifizierung\\_node.html](https://www.bsi.bund.de/EN/Themen/Oeffentliche-Verwaltung/Elektronische-Identitaeten/Online-Ausweisfunktion/eIDAS-Notifizierung/eidas-notifizierung_node.html)

<sup>5</sup> [https://www.bsi.bund.de/EN/Themen/Oeffentliche-Verwaltung/Elektronische-Identitaeten/Online-Ausweisfunktion/eID-Infrastruktur/eid-infrastruktur\\_node.html](https://www.bsi.bund.de/EN/Themen/Oeffentliche-Verwaltung/Elektronische-Identitaeten/Online-Ausweisfunktion/eID-Infrastruktur/eid-infrastruktur_node.html)

*Authorization CA* as well. For more information on how to get in contact with the *Authorization CA*, see the [German eIDAS Middleware on the German Country Page](#)<sup>6</sup>. The *Authorization Certificate* is presented to the German eID Card to ensure the authenticity of the *eID Service Provider* and to verify that the *eID Service Provider* only requests attributes allowed by the *Authorization CA*.

There are two different Authorization CAs available. The Bundesdruckerei provides Authorization Certificates for productive environments with real eID cards. Governikus provides Authorization Certificates for test environments with test eID cards.

---

<sup>6</sup> <https://eidas.ec.europa.eu/efda/browse/notification/eid-chapter-contacts/DE>

# 3

## Introduction to Docker

In addition to running the Middleware in the provided virtual machine or running the stand-alone Spring Boot application on your own server, we provide a Docker Image to run the Middleware in a Docker container.

In order to keep the container as stateless as possible, we use volumes to store the configuration and the HSQL database outside of the container.

Please note that it is not possible to run multiple Middleware containers using the same configuration or database volume.

For information on running Docker containers, see the [Docker Docs](#)<sup>7</sup>.

We provide the Docker Image:

1. [governikus/eidas-middleware-application](#) ([Docker Hub](#)<sup>8</sup>)

Two volumes are necessary to run the Middleware:

1. eidas-configuration
2. eidas-database

You can create these named volumes with the following command:

```
docker volume create eidas-configuration  
docker volume create eidas-database
```

To reduce the hassle of executing long commands in the terminal, we also provide Docker Compose files in addition to the Docker Images. This way the configuration for running the containers can be stored in configuration files. For more information on Docker Compose, see the [Docker Compose Docs](#)<sup>9</sup>.

<sup>7</sup> <https://docs.docker.com/engine/reference/run/>

<sup>8</sup> <https://hub.docker.com/r/governikus/eidas-middleware-application>

<sup>9</sup> <https://docs.docker.com/compose>

## Introduction to the VirtualBox Image

This section illustrates the requirements for the operational environment, including network configuration, DNS configuration and firewall.

### 4.1 First time login via console

In order to configure network you have to login via console first. Use `eidasmw` as username and `Pleasechangeme!` as password. To change the system settings, you will have to use the `sudo` command. Please reboot the system after configuration according to your environment and login via ssh.

### 4.2 Regenerate the SSH server key

The virtual machine is shipped without SSH server keys. You must generate new keys before using the SSH server. To generate new server keys execute the following commands:

```
sudo dpkg-reconfigure openssh-server  
sudo /etc/init.d/ssh restart
```

### 4.3 Setting up network access

The network configuration is done in the file

`/etc/network/interfaces`

The default is configured to use DHCP. It is recommended to use a static configuration in your environment. The file looks like:

```
# This file describes the network interfaces available on your system  
# and how to activate them. For more information, see interfaces(5).  
  
source /etc/network/interfaces.d/*  
  
# The loopback network interface  
auto lo  
iface lo inet loopback
```

(continues on next page)

(continued from previous page)

```
# The primary network interface
allow-hotplug enp0s3
iface enp0s3 inet dhcp
```

We advise to remove the last line and instead add a block like:

```
iface enp0s3 inet static
    address 1.1.1.2
    netmask 255.255.255.0
    gateway 1.1.1.1
```

and change the values to your specific setup.

- **address:** the IP address of this server
- **netmask:** the netmask of the used network
- **gateway:** the IP of the default gateway in this network segment

## 4.4 DNS configuration

The DNS configuration is done in the file

`/etc/resolv.conf`

The default values will probably not work in your environment! Change the following with a text editor like `vi` to your own values.

- **domain:** name of your network domain, or comment it using `#` if not applicable
- **search:** name of your network domain, or comment it using `#` if not applicable
- **nameserver:** IP address of your name server(s), use multiple `nameserver` lines if there is more than one

## 4.5 Firewall configuration

The firewall is preconfigured, all incoming connections, except the pre-configured, are denied. The settings can be found in this file:

`/etc/network/if-pre-up.d/iptables`

Pre-configured Ports:

- **ssh:** (TCP 22)
- **http:** (TCP 8080)
- **https:** (TCP 443, TCP 8443 and TCP 10000)
- **dhcp:** (UDP 67 and UDP 68)
- **snmp:** (UDP 161)

 **Hint**

Any outgoing and related or established connection is allowed. To see current firewall setup type `iptables -L -n` as root.

## Requirements

There are several requirements that have to be met in order to run the Middleware successfully. While some of them are automatically fulfilled when using the virtual machine image or Docker Image, prepare your environment according to the following requirements if you choose to just use the eIDAS Middleware JAR file.

### 5.1 Software Requirements

The eIDAS Middleware is a Spring Boot Application. This means that you can start the application with a JRE and there is no need for an application server. We support the latest available Zulu JRE in Version 17, at the time of this release this is 17.0.14.

For the operating system we support Debian 12 “Bookworm” (LTS).

### 5.2 Hardware Requirements

For productive environments, we recommend the following requirements:

- 4 CPU cores with at least 2 GHz
- 16 GB of RAM
- 100 GB of file system space

For test environments, we recommend the following requirements:

- 2 CPU cores with at least 2 GHz
- 4 GB of RAM
- 10 GB of file system space

### 5.3 Network Requirements

The eIDAS Middleware communicates with different parties. Therefore a number of ports must be open and some external URLs must be reachable.

For a **test system**, the following URLs need to be reachable:

```
https://dvca-r1.governikus-eid.de/gov_dvca/ta-service-140  
https://dvca-r1.governikus-eid.de/gov_dvca/ri-service  
https://dvca-r1.governikus-eid.de/gov_dvca/pa-service-140
```

```
http://www.bsi.bund.de/test_casca_crl  
http://download.gsb.bund.de/BSI/crl/TEST_CSCA_CRL.crl
```

For a **production system**, the following URLs need to be reachable:

```
https://berca-p1.d-trust.net/ps/dvca-at/v1_4  
https://berca-p1.d-trust.net/ps/dvsd_v2/v1_1  
https://berca-p1.d-trust.net/ps/scs/v1_4
```

```
http://www.bsi.bund.de/casca_crl  
http://download.gsb.bund.de/BSI/crl/DE_CRL.crl
```

Additionally, the eIDAS Middleware provides a port to listen for incoming eIDAS authentication requests as well as the communication with the user's web browser and eID client. While you can choose if you want to enable HTTPS in the eIDAS Middleware itself or use a reverse proxy to handle HTTPS, the aforementioned port must be open for incoming requests.

## 5.4 HSM support

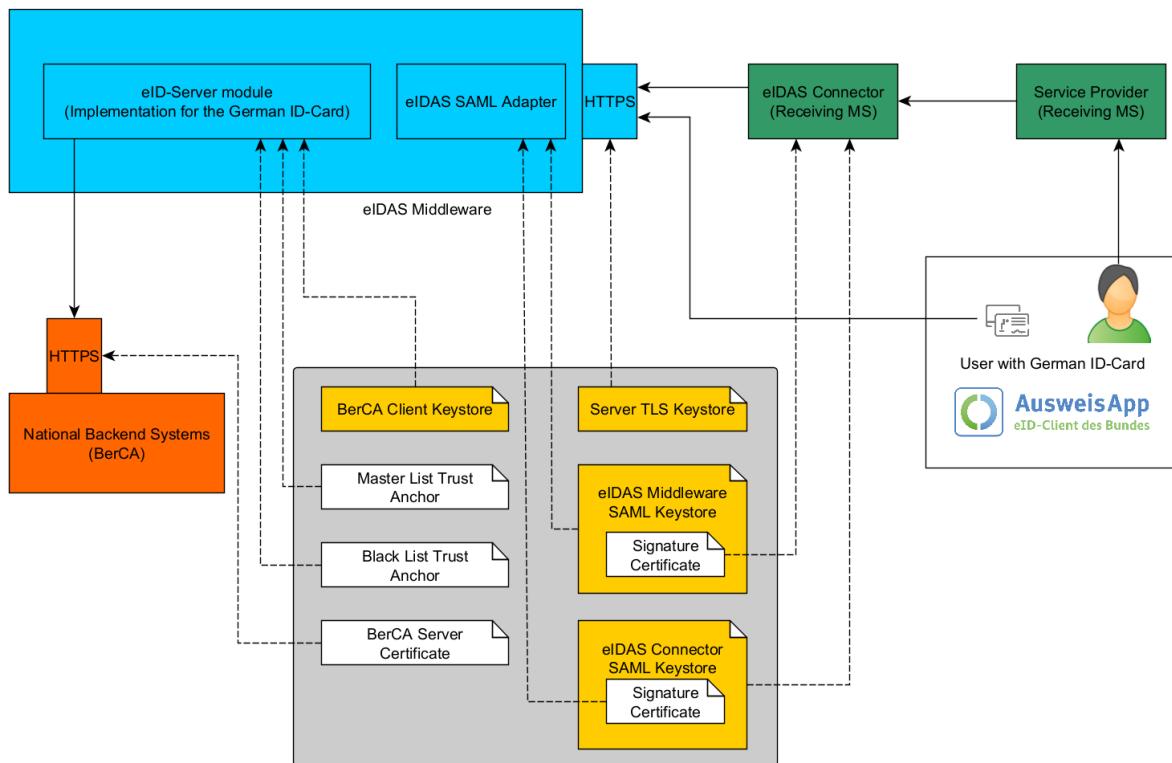
As an optional feature, the eIDAS Middleware supports storing the most important private keys in HSM instead of config files or databases. The Middleware uses the PKCS#11 standard to communicate with the HSM, which means that it should be able to function with any PKCS#11 compliant HSM. However this cannot be guaranteed since we are unable to test every single HSM model. We have successfully tested with SoftHSM.

# 6

## Configuration of the eIDAS Middleware application

### 6.1 Necessary keys and certificates

In order to setup the Middleware you will need to provide some key stores and certificates. The following overview illustrates the required certificates and key stores.



The following table describes the individual key stores and certificates:

Name	Description	Where to configure
Server TLS Key Store	This key store is used to setup the HTTPS port of the server.	application.properties > server.ssl.key-store
eIDAS Middleware SAML Key Store and Signature Certificate	This key store is used for the SAML communication with the eIDAS Connector, especially to sign the outgoing SAML responses.  In addition this key store is used to sign the SAML metadata of the eIDAS Middleware.  The corresponding certificate must be available to the remote party (eIDAS Connector) so that the SAML metadata can be verified.	Key Store: Admin-UI > eIDAS > Signature key pair Certificate: In eIDAS Connector
eIDAS Connector SAML Key Store and Signature Certificate	This key store is used for the SAML communication with the eIDAS Middleware, especially to sign the outgoing SAML requests.  In addition this key store is used to sign the SAML metadata of the eIDAS Connector.  The corresponding certificate must be available to the remote party (eIDAS Middleware) so that the SAML metadata can be verified.	Key Store: In eIDAS Connector Certificate: Admin-UI > Connector metadata > Metadata signature verification certificate
BerCA Client Key Store	This key store is needed to access the <i>Authorization CA</i> .	Admin-UI > eID service provider > DVCA client authentication key pair
BerCA Server Certificate	This certificate is needed to validate the BerCA server certificate of the <i>Authorization CA</i> .	Admin-UI > DVCA > DVCA server certificate
Master List Trust Anchor	This certificate is needed to verify the signature of the <i>Master List</i> .	Admin-UI > DVCA > Master List Trust Anchor
Block List Trust Anchor	This certificate is needed to verify the signature of the <i>Block List</i> .	Admin-UI > DVCA > Block List Trust Anchor

Please use only JKS or PKCS#12 key stores.

It is required to create the BerCA Client Key Store in consultation with the respective *Authorization CA* as they might have additional requirements for the key. They will also provide you with their TLS server certificate which needs to be entered into the configuration as well.

Be advised that the Common Name or Subject Alternative Name of the TLS Certificate must match with the URL of the Middleware as it is reachable from the internet. This is important as the AusweisApp will check the URL in the authorization certificate against the URL that is received from the eIDAS Middleware. E.g., if the Middleware is running on <https://your.eidas.domain.eu/eidas-middleware/> or <https://your.eidas.domain.eu:8443/eidas-middleware/>, the CN or SAN of the TLS certificate must include `your.eidas.domain.eu`.

For a test system, this TLS certificate may be self signed. However for a production system, this TLS certificate must meet the requirements of the [eIDAS Crypto Requirements, section 2.4](#)<sup>10</sup>.

<sup>10</sup> <https://ec.europa.eu/digital-building-blocks/sites/download/attachments/467109280/eIDAS%20Cryptographic%20Requirements%20-%20Section%202.pdf>

For a successful connection of the eIDAS Middleware with your eIDAS Connector(s), their SAML metadata must be exchanged. The metadata of eIDAS Nodes contain the certificates that should be used for encryption and signature verification, the URLs for receiving SAML requests and responses and other SAML and eIDAS related data. While it is also possible to get the metadata of the eIDAS Middleware without a signature, you must use the certificate of eIDAS Middleware SAML Signature Key Store to validate the metadata of the eIDAS Middleware. Additionally, you must provide the certificate of the eIDAS Connector SAML Signature Key Store so that the Middleware can validate the metadata of your Connector.

## 6.2 Setup the application.properties

The `application.properties` file is the main configuration file for a Spring Boot application. It must be setup prior to the first start of the application and placed in a `config` directory. The `config` directory must be in the same directory as the `eidas-middleware.jar` file, For example `/opt/eidas-middleware/config`.

Adjust the template located in `/opt/eidas-middleware/config`: (If there is no template, copy the following one)

```
#  
# Copyright (c) 2020 Governikus KG. Licensed under the EUPL, Version 1.2 or  
→as soon they will be approved by  
# the European Commission - subsequent versions of the EUPL (the "Licence");  
→You may not use this work except  
# in compliance with the Licence. You may obtain a copy of the Licence at:  
# http://joinup.ec.europa.eu/software/page/eupl Unless required by applicable  
→law or agreed to in writing,  
# software distributed under the Licence is distributed on an "AS IS" basis,  
→WITHOUT WARRANTIES OR CONDITIONS  
# OF ANY KIND, either express or implied. See the Licence for the specific  
→language governing permissions and  
# limitations under the Licence.  
#  
#server settings  
server.port=8443  
server.adminInterfacePort=10000  
#TLS settings  
server.ssl.key-store=file:/config/keystore.p12  
server.ssl.key-store-password=123456  
server.ssl.key-password=123456  
server.ssl.keyStoreType=PKCS12  
server.ssl.keyAlias=alias  
#database connection  
spring.datasource.url=jdbc:hsqldb:file:/opt/eidas-middleware/database/eidas-  
→middleware-db;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE  
spring.datasource.username=<username>  
spring.datasource.password=<password>  
#logging  
logging.file.name=/var/log/eidas-middleware/eidas-middleware.log
```

(continues on next page)

(continued from previous page)

```
#HSM
hsm.type=NO_HSM
#hsm.keys.delete=30
#hsm.keys.archive=false
#pkcs11.config=
#pkcs11.passwd=123456
blocklist.storage-folder=block-list-data
```

This configuration file contains the following sections:

### 1. Server settings

You need to provide two ports on which the Middleware will be accessed. The first (`server.port`) is the port for the eID functionality, the second (`server.adminInterfacePort`) facilitates access to the administration interface. Both ports must be given and must not contain the same value. The TLS settings are used for both ports, e.g. both ports use HTTPS or both ports use HTTP.

 **Note**

The eID port must be reachable by the users' eID clients from the internet while you can limit access to the second port to your administrators only.

### 2. TLS settings

To configure the TLS connection with your Server TLS Key Store, insert the appropriate values in this section.

 **Hint**

The TLS certificate is entangled with your [CVC](#). If the TLS certificate changes, the CVC becomes unusable until a new [CVC](#) with the correct entanglement is issued. The process of the entanglement and requesting a new [CVC](#) is automated if a [Request Signer Certificate](#) is in use (see *automatic entanglement*). Otherwise if you change the TLS key store for the eIDAS Middleware, you must inform the [Authorization CA](#) about the new TLS certificate. If you use a TLS key store that is not known to the [Authorization CA](#), the eIDAS Middleware may not work properly. If you change the TLS key store, please send an e-mail with the new TLS certificate and the CHR of your [CVC](#) data to [eidas-middleware@governikus.de](mailto:eidas-middleware@governikus.de). The CHR can be found in the admin interface on the detail page of your provider. Once the new TLS certificate is stored in the [Authorization CA](#), you will receive a reply and you can renew your [CVC](#).

### 3. Database connection

The default settings in this connection should be sufficient for most users. However you can change the database location, user name and password.

### 4. Logging

The default location for the log files is `/var/log/eidas-middleware/eidas-middleware.log`.

### 5. HSM

Set the `hsm.type` to `PKCS11` in case you want to use an HSM, or `NO_HSM` to use none. The other settings are only relevant if you use an HSM:

You need to provide a configuration file for the Sun PKCS#11 provider. In this file, you need to configure the settings for your HSM model which is out of scope of this documentation. You can find assistance for the settings in the [PKCS11 Reference Guide<sup>11</sup>](#). Then, the path to the configuration file must be given as `pkcs11.config` property

You must also provide the login password for the HSM (default user, not SO) as `pkcs11.passwd`. It is assumed that this account already exists when you start the Middleware, so you need to initialize the HSM beforehand.

You can optionally enter a period (in days) after which expired keys are deleted from the HSM. Use `hsm.keys.delete`. If you do not enter a value, a default of 30 days is assumed. Also, you can set whether you want to backup these keys in the database before they are deleted from the HSM via the `hsm.keys.archive` property. This option might not work with every HSM however.

### 6. Block List storage

Beginning in version 3.3.0, the Block List is no longer stored in the database but as text files in the file system. The folder can be configured using the property `blocklist.storage-folder`. If it is not set, the default folder is `block-list-data` (as a relative path to the eIDAS Middleware main folder). The MW requires permission to write and read in that folder.

## 6.3 Startup

Once you have the `application.properties` configured and the HSM up and running (the latter only applicable if you configured the use of an HSM), you can start the Middleware application. (see [Startup and shutdown](#))

## 6.4 Using the admin interface

As of eIDAS Middleware release 3.0.0, configuration and administration is mostly handled via web interface. Only the settings are excluded, which must be given before application startup, i.e. in the `application.properties`.

After the application has started, the admin interface is reachable with a web browser at `https://<YOUR_SERVERURL>:<YOUR_ADMIN_PORT>/admin-interface`.

If it is the first time the application is started, you will be asked to set a password to protect access to the admin interface. The eIDAS Middleware will create a `password.properties` file in the config directory. In order for the eIDAS Middleware to be able to create the file, there must be a config directory within the working directory. The working directory is the directory from which the eIDAS Middleware .jar is started, for example `/opt/eidas-middleware/config`. If you ever forget this password, you can reset it by deleting the file `password.properties` in the config directory.

The admin interface offers the following configuration options:

### 6.4.1 Import/Export configuration

Here, you can download the existing configuration as an XML file, with or without inclusion of private keys.

<sup>11</sup> <https://docs.oracle.com/en/java/javase/11/security/pkcs11-reference-guide1.html>

If you already have an XML configuration file, it is possible to upload it here, but please use this with caution as it will replace the existing configuration without further notice.

We provide XML samples for both the test and production environment, which can be uploaded and then completed using the web interface.

The XML samples for the test environment is available for download here: [Samples Test Environment](#)<sup>12</sup>

The XML samples for the production environment is available for download here: [Samples Production Environment](#)<sup>13</sup>

#### Attention

The upload function is not designed to handle POSeIDAS.xml from older Middleware versions due to changed data structures. It can deal with exports from version 3.0.0+ only.

### 6.4.2 Key management

This section deals with the management of keys and certificates. All key pairs and certificates you intend to use must be uploaded and assigned a name for further reference here.

For uploading a key pair, you first need to upload the key store containing it. Then extract the key pair from the key store. Certificates can be uploaded directly or via key store and extraction.

Deletion of key pairs and certificates is also possible. However the application will refuse if the key or certificate is still in use at some point in the configuration.

#### Attention

For PKCS#12 key stores, it is expected that they follow the convention of having key passwords identical to the key store password.

### 6.4.3 eID service provider

This section is for management of *eID Service Providers*. Each entry requires a unique name and a unique client authentication key pair.

The name is used for identifying the *eID Service Provider*. In case the *eID Service Provider* is dedicated for a private sector eIDAS SP it is imperative that the name matches the `requesterId` used in eIDAS SAML requests made by that SP. If no match is found via the `requesterId`, a second check is carried out via the `providerName`.

The client authentication key pair is used for the communication to the *Authorization CA*. The associated certificate must be given to the *Authorization CA*. In case you use a PKCS#11 HSM, this key must be stored in the HSM. It is required that the the label and the ID for the certificate and key entry in the HSM are identical. As the ID is a hexadecimal value, use the hex-value of the ASCII string.

<sup>12</sup> [https://github.com/Governikus/eidas-middleware/blob/master/doc/source/chapter/eIDAS\\_Middleware\\_configuration\\_test.xml](https://github.com/Governikus/eidas-middleware/blob/master/doc/source/chapter/eIDAS_Middleware_configuration_test.xml)

<sup>13</sup> [https://github.com/Governikus/eidas-middleware/blob/master/doc/source/chapter/eIDAS\\_Middleware\\_configuration\\_prod.xml](https://github.com/Governikus/eidas-middleware/blob/master/doc/source/chapter/eIDAS_Middleware_configuration_prod.xml)

#### 6.4.4 eIDAS

In this area, the settings for the Middleware as an eIDAS node are made. You can fill in the information that will be published in the metadata and select the service provider that will be used for requests from the public sector. Especially important are the server URL, which must have the value as the Middleware is reachable from the internet, and the SAML key pair. In case you use a PKCS#11 HSM, the key for SAML signatures must be available in the HSM using label and ID `samlsigning`. As the ID is a hexadecimal value, use the hex-value of the ASCII string.

#### 6.4.5 eID means

This section allows to change the eID means accepted by the Middleware. Please do not change this unless asked to do so.

#### 6.4.6 Timer

Here, you can set the frequency for several background jobs. Normally the default settings should suffice.

#### History

There is a history for each timer. The history shows the time and result of the last 50 executions. This is used for traceability. If a timer performs an action for several service providers, the result is displayed individually for each service provider.

#### 6.4.7 DVCA

Here, the settings for communication with the *Authorization CA* (aka DVCA) are made. For each *Authorization CA* you need to create one configuration entry and later refer to the entries in the respective service provider configurations. The values for a single DVCA configuration (certificates and URLs) can be requested from the DVCA operator.

When configuring a new eIDAS Middleware, the certificate for the Master List trust anchor must be the currently used Master List signer certificate or its issuer. After a successful update of the Master List, the Master List itself acts as an additional trust anchor for future Master List updates. Therefore, the Master List trust anchor does not need to be updated and it is fine to keep this certificate in the configuration, even if it expires.

##### 💡 Hint

In case the Master List cannot be updated because no valid trust anchor is present, contact the German POSC or the Governikus eIDAS Middleware support to get the current Master List trust anchor.

#### 6.4.8 Connector metadata

In this area, you can configure the metadata files of eIDAS nodes which will connect to the Middleware, as well as the certificate to check the metadata signatures.

## Operating the server

### 7.1 Startup and shutdown

The commands for starting and stopping vary between the different environments.

#### 7.1.1 Plain eIDAS Middleware JAR

If you choose to use just the JAR in your own environment, execute the following commands to start the eIDAS Middleware.

```
# cd into the folder where the eidas-middleware.jar is located in  
java -jar eidas-middleware.jar
```

If the configuration files are not located in a subdirectory called config of your current working directory, you must specify the location of the configuration directory. Note that the path must end with a /:

```
java -jar eidas-middleware.jar --spring.config.additional-location=file:/path/  
→to/your/configuration-directory/
```

To stop the eIDAS Middleware execute CTRL+C. Alternatively you can also send SIGTERM with kill [PID of the Middleware].

Based on these commands you can write your own scripts for starting and stopping.

#### 7.1.2 VirtualBox Image

In the image you can execute the same commands as for the plain eIDAS Middleware JAR. The JAR file is located in /opt/eidas-middleware.

In addition to starting the Middleware directly, you can use systemd to start and stop the application.

```
sudo systemctl start eidas-middleware.service  
sudo systemctl stop eidas-middleware.service
```

To start the application on boot, execute the following line.

```
sudo systemctl enable eidas-middleware.service
```

### 💡 Hint

The eidas-middleware.service contains hard coded paths for the java installation and the name of the application. When you update or rename your java installation or the application, you must adapt the paths in the service file.

### 7.1.3 Docker

The configuration and database of the Middleware are located in named volumes. This way you can stop and remove the Middleware Docker container and create a new one to restart the application.

To run the eIDAS Middleware, execute the following command. It will mount the named volumes containing the database and configuration in the container and the application will be available on port 8443.

```
docker run --rm -it -v eidas-configuration:/opt/eidas-middleware/
  ↪configuration -v eidas-database:/opt/eidas-middleware/database -p 8443:8443
  ↪--name eidas-middleware-application governikus/eidas-middleware-
  ↪application:4.0.1
```

To stop and remove the container, just hit CTRL+C.

To keep the container running longer without being attached to the STDOUT and STDERR, change the command to the following:

```
docker run -d -v eidas-configuration:/opt/eidas-middleware/configuration -v
  ↪eidas-database:/opt/eidas-middleware/database -p 8443:8443 --name eidas-
  ↪middleware-application governikus/eidas-middleware-application:4.0.1
```

For more information on starting and stopping containers and viewing the logs, see the [Docker Docs](#)<sup>14</sup>.

As mentioned before, the eIDAS Middleware application configuration is located in the named volume. If you want to change the TLS key store or disable HTTPS because you are using a reverse proxy, you should use the admin interface to modify the configuration instead of adding environment variables to the Docker run command.

To use this container with Docker Compose, see the example Docker Compose file at [GitHub](#)<sup>15</sup>. You can use docker-compose.yaml out of the box to start the eIDAS Middleware application.

```
cd eidas-middleware/docker-compose
docker-compose up
```

To stop the container, hit CTRL+C. To remove the container afterwards, execute docker-compose down.

## 7.2 Obtain authorization certificate

When the Middleware is running, direct your browser to the admin interface at [https://<YOUR\\_SERVERURL>:<YOUR\\_ADMIN\\_PORT>/admin-interface](https://<YOUR_SERVERURL>:<YOUR_ADMIN_PORT>/admin-interface).

After logging in, you will see your *eID Service Providers* in the dashboard (assuming you have already completed the configuration as in [Configuration of the eIDAS Middleware application](#)). Please open the details for the provider you want to obtain the certificate for.

<sup>14</sup> <https://docs.docker.com/engine/reference/run/>

<sup>15</sup> <https://github.com/Governikus/eidas-middleware/blob/master/eidas-middleware/docker-compose/docker-compose.yaml>

By clicking the button `Start connection check` you can check the connection to the *Authorization CA*. If this check does not succeed, take a look in the log for more details. Possible errors are firewalls that block the connection to the *Authorization CA* or the *Authorization CA* has not yet stored your client TLS server certificate. If the error persists, send the log file and your error description to [eidas-middleware@governikus.com](mailto:eidas-middleware@governikus.com).

You are also advised to create a *Request Signer Certificate* before you send the initial request. Navigate to the tab RSC.

It is not required to specify a holder for public sector *eID Service Provider*. In this case, the RSC can be generated by pressing *Generate RSC* without specifying a holder.

For private sector *eID Service Providers*, please enter the holder for the *Request Signer Certificate* assigned to the provider by the *Authorization CA* and then press *Generate RSC*. The Holder for the private sector consists of the country code and the Holder Mnemonic. If the Holder Mnemonic is not known, please contact [eidas-middleware@governikus.de](mailto:eidas-middleware@governikus.de) to get the Holder Mnemonic.

If the request signer certificate has been successfully generated the button changes to *Download RSC*. You can download the certificate and forward it to the PoSC or its representative (production system) / *Authorization CA* (test system).

If the connection check is successful and the *Authorization CA* has confirmed that your Request Signer Certificate had been processed, you can request the *Authorization Certificate*. To do that, fill in the form *Start an initial request* with the values that you should have received from the *Authorization CA*. If the CA did not specify a sequence number, you can start with 1. Then click on *Send initial request to DVCA*. If this request was unsuccessful, take a look in the log for more details and double check that the country code and CHR Mnemonic are correct. If the error persists, send the log file and your error description to [eidas-middleware@governikus.com](mailto:eidas-middleware@governikus.com).

After a successful initial request the eIDAS Middleware should be ready to receive eIDAS requests from your eIDAS connector.

The eIDAS Middleware automatically renews the *Authorization Certificate*. It also checks regularly for updates of the *Block List*, *Master List* and *Defect List*.

## 7.3 Additional information

### 7.3.1 Logging

The log level can be changed by adding properties to the `application.properties`.

```
# change the root level:
logging.level.root=DEBUG
# change the logging level only of the Middleware specific classes:
logging.level.de.governikus=DEBUG
```

For more information, see the [Spring Boot documentation<sup>16</sup>](#).

#### 💡 Hint

Log level info is recommended for productive operation. The debug level should only be used if a problem needs to be analyzed.

<sup>16</sup> <https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-logging.html#boot-features-logging>

### 7.3.2 Startup checks

The Middleware performs some checks when it is started. In detail, these are:

- Is the TLS server certificate valid?
- Is the *Authorization Certificate* valid?
- Does the server URL match the one in the *Authorization Certificate*?
- Is the TLS server certificate correctly referenced in the *Authorization Certificate*?

The results of these checks can be found in the log. Failed checks are reported as warnings, while successful checks are logged on the info level. Also, you can see the results in the admin interface and trigger a rerun of the checks there.

#### Note

The check for TLS server certificate validity performs a call to the server URL in order to retrieve the certificate. If this call is blocked, or routed to a different point than calls originating from the internet, you may experience false negative results.

### 7.3.3 VirtualBox Image

The operating system is configured to use the official Debian sources for updates. Please make sure that updates are installed on a regular basis. To update the operating system issue the following commands:  
`apt-get update && apt-get upgrade`

When using systemd, the eIDAS Middleware log files can be found in the directory `/var/log/eidas-middleware`.

### 7.3.4 Scalability

The performance of the eIDAS Middleware improves by adding more memory (RAM) and using a faster CPU. In case the memory configuration has changed, the server needs to be restarted. To start the JVM with more memory, add `-Xmx` with the new maximum memory size to the start command, e.g. `java -Xmx8g -jar eidas-middleware-4.0.1.jar` for 8 GB.

### 7.3.5 Request Signer Certificate

The Middleware supports *Request Signer Certificates*. These are long-term certificates used to sign requests for *Authorization Certificates*, both initial and subsequent. When you have not yet generated one, you can do so by clicking *Generate RSC*. After that, the new request signer certificate will automatically be used for any *Authorization Certificate* request.

This is especially helpful to Middleware operators in case the *Authorization Certificate* expires before it has been renewed, as they can simply sign a new initial request using the *Request Signer Certificate* and do not need support by the *Authorization CA*.

However, in order to accept requests signed with the *Request Signer Certificate*, the *Authorization CA* needs to know this certificate. You need to download the freshly generated *Request Signer Certificate* using *Download RSC* and forward it to the PoSC or its representative (production system) / *Authorization CA* (test system). As a fallback, the *Authorization Certificate* requests are still signed using certificate chain method when the *Authorization CA* has not yet processed your new request signer certificate.

A *Request Signer Certificate* is always valid for 3 years.

The current *Request Signer Certificate* can be downloaded anytime by clicking the “Download RSC” button.

The current *Request Signer Certificate* can be deleted by clicking the “Delete Current RSC” button.

### Automatic renewal of the request signer certificate

The Middleware can automatically send new *Request Signer Certificate* to the *Authorization CA*. For security reasons, automatic renewal can only be performed if a *Request Signer Certificate* is already registered with the *Authorization CA*.

A message is displayed in the admin interface if the *Request Signer Certificate* must be sent manually to the *Authorization CA*. As soon as a *Request Signer Certificate* is used by the Middleware, a green tick “Is RSC in use:” is displayed in the Admin interface under the “RSC” tab in the “RSC Info” tile in the eID Service Provider segment.

A timer checks daily whether the existing *Request Signer Certificate* needs to be renewed. By default, a *Request Signer Certificate* is renewed 3 weeks before it expires and the new *Request Signer Certificate* is automatically sent to the *Authorization CA*.

If the renewal fails, a message is displayed in the administration interface. The value when a *Request Signer Certificate* should be renewed can be adjusted in the timer configuration.

### Manual renewal of the request signer certificate

*Request Signer Certificate* can also be renewed and sent to the *Authorization CA* manually via the admin interface. In the Admin interface, there is a ‘Generate And Send RSC’ button under the ‘RSC’ tab in the eID Service Provider area. By clicking on this button, a new *Request Signer Certificate* is generated and sent to the *Authorization CA*.

If the process was not successful, an error is displayed in the admin interface. If a new pending *Request Signer Certificate* was created but not successfully sent to the *Authorization CA*, an attempt can be made to resend the pending *Request Signer Certificate* to the *Authorization CA*. Please check the logs first or contact [eidas-middleware@governikus.de](mailto:eidas-middleware@governikus.de) to find out which problem is preventing the renewal.

If you want to delete the pending RSC, click on the “Delete Pending RSC” button.

### Automatic renewal of expired authorization certificates

In case an *Authorization Certificate* has expired, a new *Authorization Certificate* can be requested automatically with a request signer certificate. As long as the expiration date of the *Authorization Certificate* is not more than two days ago, an attempt is made every six hours to request a new *Authorization Certificate* with the request signer certificate. The message “*The CVC has expired. The system tries to renew the CVC in the background. Alternatively, a manual initial request can be performed*” will be visible in that timespan.

### Automatic entanglement after TLS server certificate change

To work properly, an *Authorization Certificate* must have the hash of the TLS server certificate of this eIDAS Middleware in its description. This link between the *Authorization Certificate* and the TLS server certificate is called entanglement. The eIDAS Middleware checks in a configurable interval the entanglement of the servers TLS server certificate with the *Authorization Certificate* of each service provider. If the following preconditions are met, the eIDAS Middleware entangles a new TLS server certificate and renews the *Authorization Certificate* automatically:

- The timer for automatic entanglement is activated.

- The service provider is enabled.
- The service provider has an *Authorization Certificate*.
- The service provider has a *Request Signer Certificate*.
- The *Request Signer Certificate* of the service provider is known to the *Authorization CA*.
- The eIDAS Middleware can reach its own configured URL and a TLS server certificate is found.
- The found TLS server certificate is not expired.
- The found TLS server certificate is not linked to the current *Authorization Certificate*.

### Automatic renewal of TLS client certificate

The Communication between the *Authorization CA* and the eIDAS Middleware is secured by TLS Client Authentication. The TLS client certificate of the eIDAS Middleware is valid for 3 years. Starting from version 3.3.0, the *Authorization CA* offers an interface to automatically renew a TLS client certificate. To use this service a valid *Request Signer Certificate* is mandatory, as the request to the *Authorization CA* is signed with a *Request Signer Certificate* and the *Authorization CA* checks the signature with the *Request Signer Certificate* stored in the *Authorization CA*. The use of an *Request Signer Certificate* is described in section 7.3.5.

The renewal of a TLS client certificate consists of two calls to the *Authorization CA*. The first request is a certificate signing request (CSR) which is sent to the *Authorization CA*. The *Authorization CA* checks the CSR and provides a signed certificate, which will be pulled via a second request. The eIDAS Middleware checks daily whether the current TLS client certificate is valid for less than 30 days. If this is the case, a CSR is automatically generated and sent to the *Authorization CA*. If a CSR has already been sent, the system checks whether the new TLS client certificate can be retrieved from the *Authorization CA*. If it is successfully retrieved, the new TLS client certificate is automatically stored in the configuration and used for future connections to the *Authorization CA*.

A new key pair must be used to sign the CSR which will be generated by the eIDAS Middleware and used for the CSR. As soon as the new certificate has been successfully retrieved from the *Authorization CA*, the certificate will automatically be stored in the configuration and used for the connection to the *Authorization CA*. The value for when a TLS client certificate should be renewed can be configured via the Admin UI under Timer Configuration. The default value is 30 days.

The TLS client certificate can also be renewed manually via the Admin UI. In the Admin UI, you will find a “TLS” tab in the “Service Provider Details” section. This page lists the expiry date of the current TLS client certificate and, if a CSR already exists, the date from which the new TLS client certificate can be pulled from *Authorization CA*. On the right-hand side, a button is available to generate a new CSR and send it to the *Authorization CA*. If no HSM is used, a drop-down menu is shown above the “Generate and Send CSR” button in which the key to be used for signing the CSR can be selected. The ‘Generate New Key Pair’ is preset. If the eIDAS Middleware should generate a new key pair, ‘Generate New Key Pair’ must be selected in the drop-down menu. It is recommended to select ‘Generate New Key Pair’ so the security requirements are met. It is also possible to upload a key via the Key Management. The key must be an RSA key with at least 3072 bits length.

If an HSM is used, two radio buttons are available. You can select whether a new key pair should be generated or a present key should be used. If a present key should be used, it must be generated by the operator beforehand, using the “CVCRefID” of the service provider plus the suffix “-PendingTLS” as a label, for example “providerA-PendingTLS”. The CVCRefID of the service provider can be found in the XML configuration of the eIDAS Middleware, which can be downloaded via the Admin UI on the import/export configuration page. This allows the operator to generate a key in the HSM that should be used. Please be aware that currently only RSA keys with at least 3072 bits are supported.

As soon as a CSR has been created and successfully received by the *Authorization CA*, the new TLS client certificate can be pulled via the Admin UI. The information on when the new TLS client certificate is available for collection is displayed on the left-hand side by the date and time ‘Not Poll Before’. When the time has been reached from which the TLS client certificate can be collected, the new TLS client certificate can be pulled by clicking the ‘Poll Client Certificate’ button. If the new TLS client certificate has been successfully received, the new TLS client certificate is automatically activated in the configuration and will be used for the connection to the *Authorization CA*. It is also possible to delete a pending CSR by clicking the button ‘Delete Pending CSR’.

### Attention

Please note that a manual renewal should only be performed after careful consideration or at the request of the support team.

## 7.4 Monitoring

### 7.4.1 SNMP Agent (system)

The virtual machine we provide has a system *SNMP* agent enabled by default and preconfigured. With this agent, you can monitor the health status of the server using the SNMP tools of your choice. You can change the SNMP settings by editing the file `/etc/snmp/snmpd.conf`.

The configured user name is `gov` with authentication protocol SHA and privacy protocol DES, both passwords `12345678`.

For example, a `snmpwalk` on OID `1.3.6.1.2.1.25.4.2.1.4` (`HOST-RESOURCES-MIB::hrSWRunPath`) will reveal the running processes.

```
snmpwalk -v3 -l authPriv -u gov -a SHA -A 12345678 -x DES -X 12345678
$HOSTNAME 1.3.6.1.2.1.25.4.2.1.4
```

Check the output on whether it contains `java`.

You can monitor CPU, network and memory consumption with the usual OIDs, see [SNMP Documentation<sup>17</sup>](#) for details.

The following example will show the total RAM usage:

```
snmpget -v3 -l authPriv -u gov -a SHA -A 12345678 -x DES -X 12345678 $HOSTNAME
1.3.6.1.4.1.2021.4.6.0
```

### 7.4.2 SNMP Agent (application)

The Middleware application itself also has an internal SNMP agent which can be used to query various data directly from the application (see below for what is available). Please note this is a different agent than the one for the system. The system agent is only preconfigured in the virtual machine, the internal agent is available directly in the Middleware application. If you intend to use both in parallel, you have to make sure that they run on different ports.

To activate the internal SNMP agent it is necessary to set `poseidas.snmp.username`, `poseidas.snmp.authpwd` (authentication password) and `poseidas.snmp.privpwd` (encryption password) in the `application.properties`. The passwords have a minimum length of 8 characters. The SNMP agent supports GET and GET NEXT requests.

<sup>17</sup> <http://www.debianadmin.com/linux-snmp-oids-for-cpumemory-and-disk-statistics.html>

Optional properties are `poseidas.snmp.authalgo` (authentication algorithm) with one of these values: md5, sha, hmac128sha224, hmac192sha256, hmac256sha384, hmac384sha512 (hmac384sha512 is the default value when not set), `poseidas.snmp.privalgo` (encryption algorithm) with one of these values: des, 3des, aes128, aes192, aes256 (aes256 is the default value when not set).

There are two different ways to use the SNMP agent to monitor the application. It is divided in GET and TRAP.

Optional properties for GET and GET NEXT requests can be set in the application.properties: `poseidas.snmp.agenthost` with the default value set to localhost and `poseidas.snmp.agentport` with the default value set to port 161.

For monitoring the TRAPs it is also necessary to set `poseidas.snmp.managementhost` in the application.properties.

Optional property for TRAP is `poseidas.snmp.managementport` (port 162 is the default value when not set).

All existing SNMP GET values are explained in detail in the MIB located at <https://github.com/Governikus/eidas-middleware/blob/4.0.1/poseidas/snmp/EIDASMW-SNMP-MIB.mib>.

OID	GET (Return datatype)	value	Description
1.3.6.1.4.1.28939.3.1.1.5.2	lastSuccessfulCRLRetrieval (DateAndTime)		The timestamp for the last successful retrieval of a certificate revocation list is returned
1.3.6.1.4.1.28939.3.1.1.5.1	isCRLAvailable (Integer32)		0: No certificate revocation list is present, 1: A certificate revocation list is present
1.3.6.1.4.1.28939.3.1.1.11	tlsCertificateExpirationDate (DateAndTime)		Expiration date of the server certificate

Governikus OID = 1.3.6.1.4.1.28939. eIDAS Middleware prefix = 3. Get prefix = 1.

OID	GET (Return datatype)	Description
1.3.6.1.4.1.28939.3.1.2.1.1	serviceProviderName (OCTET STRING)	The service provider name used for identifying instances of the columnar objects in the service-ProviderTable
1.3.6.1.4.1.28939.3.1.2.1.11	cvcPresent (Integer32)	CVC present: 0 = not present, 1 = present
1.3.6.1.4.1.28939.3.1.2.1.12	cvcExpirationDate (DateAndTime)	Date until the CVC is valid
1.3.6.1.4.1.28939.3.1.2.1.13	cvcSubjectUrl (OCTET STRING)	The Subject URL of the CVC
1.3.6.1.4.1.28939.3.1.2.1.14	cvcAndTlsLinked (Integer32)	TLS server certificate referenced in CVC: 0 = not linked, 1 = linked
1.3.6.1.4.1.28939.3.1.2.1.21	blackListAvailable (Integer32)	Block List availability: 0 = not available, 1 = available
1.3.6.1.4.1.28939.3.1.2.1.22	lastBlackListRenewal (DateAndTime)	Date of last successful Block List renewal
1.3.6.1.4.1.28939.3.1.2.1.23	blackListCAAvailable (Integer32)	Block List PKI availability: 0 = not available, 1 = available
1.3.6.1.4.1.28939.3.1.2.1.31	masterListAvailable (Integer32)	Master List availability: 0 = not available, 1 = available
1.3.6.1.4.1.28939.3.1.2.1.32	lastMasterListRenewal (DateAndTime)	Date of last successful Master List renewal
1.3.6.1.4.1.28939.3.1.2.1.33	masterListCAAvailable (Integer32)	Master List PKI availability: 0 = not available, 1 = available
1.3.6.1.4.1.28939.3.1.2.1.41	defectListAvailable (Integer32)	Defect List availability: 0 = not available, 1 = available
1.3.6.1.4.1.28939.3.1.2.1.42	lastDefectListRenewal (DateAndTime)	Date of last successful Defect List renewal
1.3.6.1.4.1.28939.3.1.2.1.43	defectListCAAvailable (Integer32)	Defect List PKI availability: 0 = not available, 1 = available
1.3.6.1.4.1.28939.3.1.2.1.61	rscPendingAvailable (Integer32)	Pending RSC availability: 0 = not available, 1 = available
1.3.6.1.4.1.28939.3.1.2.1.62	rscCurrentValidUntil (DateAndTime)	Last date of RSC validity
1.3.6.1.4.1.28939.3.1.2.1.71	tlsClientCertificate-ValidUntil (DateAndTime)	Expiry date of the TLS client certificate

The following table will show the OIDs and their meaning.

Governikus OID = 1.3.6.1.4.1.28939. eIDAS Middleware prefix = 3. Trap prefix = 2.

OID	Description	Messages (Datatype)
1.3.6.1.4.1.28939.3.2.1.1	The status of the last renewal of the CVC	0 = success, 1 = failed (Integer32)
1.3.6.1.4.1.28939.3.2.2.1	The status of the last renewal of the Block List	0 = renewed, 1 = no list received, 2 = list signature check failed, 3 = list processing error (Integer32)
1.3.6.1.4.1.28939.3.2.2.2	The last renewal processing duration of the Block List	The duration in milliseconds (long)
1.3.6.1.4.1.28939.3.2.3.1	The status of the last renewal of the Master List	0 = renewed, 1 = no list received, 2 = list signature check failed, 3 = list processing error (Integer32)
1.3.6.1.4.1.28939.3.2.3.2	The last renewal processing duration of the Master List	The duration in milliseconds (long)
1.3.6.1.4.1.28939.3.2.4.1	The status of the last renewal of the Defect List	0 = renewed, 1 = no list received, 2 = list signature check failed, 3 = list processing error (Integer32)
1.3.6.1.4.1.28939.3.2.4.2	The last renewal processing duration of the Defect List	The duration in milliseconds (long)
1.3.6.1.4.1.28939.3.2.5.1	The status of the last renewal of the Certificate Revocation List	0 = success, 1 = failed (Integer32)
1.3.6.1.4.1.28939.3.2.6.1	A pending Request Signer Certificate is now current	0 = success, 1 = failed because there is no pending rsc, 2 = failed because there is no RefID (Integer32)
1.3.6.1.4.1.28939.3.2.6.2	A new pending Request Signer Certificate has been generated	Certificate information (OCTET STRING)

## 7.5 Test mode

The eIDAS Middleware includes a test mode to demonstrate handling various errors. To do this, a RequestedAuthnContext must be added to the SAML request, e.g.

```
<saml2p:RequestedAuthnContext Comparison="minimum">
    <saml2:AuthnContextClassRef xmlns:saml2="urn:oasis:names:tc:SAML:2.
    ↵0:assertion">test</saml2:AuthnContextClassRef>
</saml2p:RequestedAuthnContext>
```

Possible values are:

```
test
test#CANCELLATIONBYUSER
test#WRONGPIN
test#WRONGSIGNATURE
test#CARDEXPIRED
```

(continues on next page)

(continued from previous page)

test#UNKNOWN

## eIDAS Demo Application

Starting with version 1.0.4, we provide a simple eIDAS Demo Application to test the eIDAS Middleware. The eIDAS Demo Application is an eIDAS Connector that sends an eIDAS request to the Middleware and displays the returned eIDAS response. This way you can verify that the eIDAS Middleware is working correctly.

### 8.1 Configuration of the eIDAS Demo Application

Because the eIDAS Demo Application is a Spring Boot Application, the `application.properties` file is used to configure the eIDAS Demo Application. It must contain the following content. Adapt all the values to match your setup.

```
# The URL to the eIDAS Middleware
middleware.samlreceiverurl=https://your.middleware.host:8443/eidas-middleware/
˓→RequestReceiver

# The path to the certificate that can be used to verify the signature of the
˓→eIDAS responses from the Middleware
middleware.signature.certificate=/path/to/your/middleware_signature.crt

# The path to the key store that will be used to sign the eIDAS request, jks
˓→or p12
demo.signature.keystore=/path/to/your/demo_signature_keystore.p12

# The alias for the demo_signature_keystore
demo.signature.alias=alias

# The pin of the keystore and key of the demo_signature_keystore
demo.signature.pin=123456

# The path to the key store that will be used to decrypt the eIDAS response,
˓→jks or p12
demo.decryption.keystore=/path/to/your/demo_decryption_keystore.p12

# The alias for the demo_decryption_keystore
demo.decryption.alias=alias
```

(continues on next page)

(continued from previous page)

```
# The pin of the keystore and key of the demo_decryption_keystore
demo.decryption.pin=123456

# Optional: Add a context path to the eIDAS Demo Application
#server.contextPath=/eIDASDemoApplication

# Optional: Change the port of the eIDAS Demo Application
server.port=8080
```

The application.properties file must be present in your working directory or in a subdirectory called config of your working directory. Similarly to the eIDAS Middleware you can use HTTPS as well. The configuration is according to the application.properties of the eIDAS Middleware.

## 8.2 Using the eIDAS Demo Application

To use the eIDAS Demo Application, start by running the eIDAS Demo Application.

1. Change to the correct directory where the aforementioned configuration is present.
2. If not present, copy the eidas-demo-4.0.1.jar file in this directory.
3. Start the application by executing `java -jar eidas-demo-4.0.1.jar`.

Now you must configure your eIDAS Middleware to communicate with the eIDAS Demo Application.

1. Open the URL `http://your.demo.host:8080/Metadata`. If you have changed the port or added a context path, adapt the URL to your configuration.
2. Right-click on the page in your browser and select `Save as` to save the Connector metadata of the eIDAS Demo Application.
3. Export the certificate from the demo\_signature\_keystore.
4. Configure the eIDAS Middleware with this metadata and certificate using the admin interface, following the instructions from [Configuration of the eIDAS Middleware application](#) but use these files when asked to upload the metadata of the connector and the corresponding signature certificate.

Once you have configured the eIDAS Middleware, you are ready to test your system.

1. Open the URL `http://your.demo.host:8080/NewRequesterServlet`.
2. Click on `Go to the eIDAS Middleware`.
3. Click on `Understood, start online identification` to start the authorization procedure with your test eID card and the AusweisApp.
4. Finally, you should be redirected to `http://your.demo.host:8080/NewReceiverServlet`.

At the top you can see the full eIDAS SAML response with the encrypted SAML assertion. Below that you can see the data from the test eID card. If there was an error or the user aborted the authorization procedure, you would see the eIDAS SAML response with the status and unencrypted assertion containing more information why the error occurred.

There is also the possibility to demonstrate the eIDAS Middleware handling various errors. Open the URL `http://your.demo.host:8080/NewRequesterServlet`. The third part of the linklist sends LoA = Test with different error provocations. This test works without eID card and AusweisApp. In this

demonstration also the CVC check is conducted. The result is shown if the CVC check wasn't successful. If the CVC check was successful the eIDAS Middleware is configured properly.

## 8.3 Using the eIDAS Demo Application in Docker

We provide a Docker Image for the eIDAS Demo Application as well. It can be found at DockerHub<sup>18</sup>.

Because the `application.properties` file and the referenced key stores must be mounted into the container, it is recommended to place the configuration file and the key stores and certificates in a single directory. Also bear in mind that you must use the path of the container file system in the configuration when setting the key stores and certificates, e.g. `middleware.signature.certificate=/opt/eidas-middleware/config/middleware_signature.crt`

To run the Middleware, execute the following command after you have prepared the configuration, certificate and key stores:

```
docker run --rm -it -v /path/to/your/config-directory:/opt/eidas-middleware/  
→config -p 8080:8080 governikus/eidas-demo-application:4.0.1
```

Now you can follow the steps above to configure and test the eIDAS Middleware.

To stop the eIDAS Demo Application, execute CTRL+C.

There is also a docker compose file available at GitHub<sup>19</sup>. You can copy this file on your local machine and create a directory called configuration in your working directory. Then copy the configuration file and the certificate and key stores in the configuration directory. Afterwards you can run the container by executing `docker-compose up`. To stop and remove the container, first execute CTRL+C followed by `docker-compose down`.

---

<sup>18</sup> <https://hub.docker.com/r/governikus/eidas-demo-application/>

<sup>19</sup> <https://github.com/Governikus/eidas-middleware/blob/master/eidas-demo/docker-compose/docker-compose.yaml>

## Changelog

- 1.0.1
  - Initial Release.
- 1.0.2
  - Improvement of the *Black List* handling.
- 1.0.3
  - Improvement of the *Black List* handling with database changes. NOTE: When upgrading from 1.0.1 or 1.0.2 to 1.0.3 or later, you must execute the database migration application.
  - Introduction of the configuration wizard.
  - Improvements for Docker integration.
  - Various bug fixes.
- 1.0.4
  - Add the eIDAS SAML Demo.
  - Fix ID generation and audience restriction in the eIDAS SAML response.
  - Fix for the eIDAS Middleware Docker integration.
  - Fix the project setup of the password generator.
  - Older databases are now also supported in the database migration tool.
  - Other minor fixes and improvements.
- 1.0.5
  - eIDAS Middleware: Fix wrong certificate in the metadata for the encryption part.
  - eIDAS Middleware: The URL for the entityID in the metadata must now be set in the configuration. Add SERVER\_URL=https://host:port to eidasmiddleware.properties.
  - Configuration Wizard: Adapt for new configuration value. Use the value from the eID configuration page.
  - Configuration Wizard: Ensure that only one connector metadata file is saved.

- Configuration Wizard: Ensure that the original filenames are used when loading and saving a previous configuration.
- 1.0.6
  - eIDAS Middleware: Fix bug introduced with version 1.0.5 where two URLs in the SAML response were switched.
  - eIDAS Middleware: Improve logging in case of unparsable authentication request.
- 1.0.7

### Security Advisory

There were two security issues reported to the German POSC and Governikus. This release fixes these issues. It is strongly recommended to immediately update to this release as the XXE attack allows an unauthenticated remote attacker to read ASCII files from the file system which can be read by the Middleware Java process.

- eIDAS Middleware: **Security Fix** Endpoints that parse XML content like /RequestReceiver or /paosreceiver were vulnerable to XXE attacks. These endpoints are no longer vulnerable against XXE attacks.
- eIDAS Middleware: **Security Fix** The /TcToken endpoint was vulnerable against XXS attacks as requests parameters were inserted in the HTML response. All endpoints that display HTML content no longer insert user input into the HTML content.
- eIDAS Middleware: The Master List Trust Anchor for the POSeIDAS\_PRODUCTION.xml template is updated.
- 1.1.0
  - eIDAS Middleware: Support multiple metadata files as well as multiple *eID Service Providers*.
  - Configuration Wizard: Support multiple metadata files and multiple *eID Service Providers*.
  - eIDAS Middleware: Implement the recommendations from the latest pen test.
  - eIDAS Middleware: The admin interface can listen on a different port than the actual application endpoints. The admin interface is now available on a different context path, e.g. [https://\[host\]:\[port\]/admin-interface/list](https://[host]:[port]/admin-interface/list)
  - eIDAS Middleware: New HTML designs for starting the AusweisApp2 and error messages.
  - eIDAS Middleware: A HSM Module can be used via the PKCS#11 interface for different cryptographic actions.
  - Configuration Wizard: Support HSM configuration.
  - eIDAS Middleware: Fix padding and OID identifier in case ECDH encryption is used.
  - eIDAS Middleware: Add whitelist for allowed document signer types which can be extended using the configuration.
  - eIDAS Middleware: The validity of the metadata is now configurable, the default value is 30 days.
- 1.1.1
  - eIDAS Middleware: Fix a bug where the newest generation of German eID cards were not accepted.

- 1.2.0
  - eIDAS Middleware: Fix handling of empty or absent RelayState.
  - eIDAS Middleware: Fix the order of attributes in the current address.
  - eIDAS Middleware: Fix some typos in the creation of SAML responses.
  - eIDAS Middleware: Remove the assertion in SAML responses when status is not success.
  - eIDAS Middleware: Remove carriage returns in the base64 representation of the SAML response.

Note: The carriage returns inside the SAML response, e.g. in signatures and cipher texts, are not removed. These are created by OpenSAML / xmlsec following W3C XML signature and encryption specifications.

- 1.2.1
  - eIDAS Middleware: Fix SAML encryption with EC certificates.
- 1.2.2
  - eIDAS Middleware: Fix a bug where the newest generation of German eID cards were not accepted.
- 1.2.4
  - eIDAS Middleware: Security Patch
- 1.2.5
  - eIDAS Middleware: Change certificate chain building algorithm.
  - eIDAS Middleware: Update xmlsec and bouncycastle.
- 1.2.7
  - eIDAS Middleware: Fix Log4j security issue.
- 1.2.8
  - eIDAS Middleware: Update log4j to version 2.17.1.
- 1.2.9
  - eIDAS Middleware: Update third party libraries for security fixes.
- 1.2.10
  - eIDAS Middleware: Security Patch
- 2.0.0
  - eIDAS Middleware: Support version 1.2 of the eIDAS specifications.
  - eIDAS Middleware: Add a CRL check to Passive Authentication.
  - eIDAS Middleware: There is no longer a dedicated Defect List trust anchor. Trust is instead established using the Master List.
  - Configuration Wizard: Remove option to configure Defect List trust anchor.
  - eIDAS Middleware: Perform some certificate checks on startup.

- eIDAS Middleware: Option to have unsigned metadata, including download button in admin interface.
- eIDAS Middleware: Overhaul admin interface.
- eIDAS Middleware: Display status of AusweisApp2 on Middleware landing page.
- eIDAS Middleware: Load resources (css, js, ...) with context path.
- eIDAS Middleware: Add support for a second test CA.
- Configuration Wizard: Option to configure second test CA.

### Note

The new test CA is introduced to slowly replace the old one. Do not change CA settings on your own. The process of phasing out the old and migrating to the new will be initiated and guided by Governikus.

- 2.0.1
  - eIDAS Middleware: Fix a bug where the newest generation of German eID cards were not accepted.
- 2.1.0
  - eIDAS Middleware: The new eID card for citizens of the European Union and the European Economic Area is automatically accepted by default.
  - eIDAS Middleware: Support for *Request Signer Certificates*.
  - eIDAS Middleware: Replace the template-based SAML message generation with Open-SAML methods.
  - eIDAS Middleware: ProviderName is treated as optional in AuthnRequests, it is independent of the RequesterID.
  - eIDAS Middleware: NameIDPolicy is treated as optional in AuthnRequests, illegal values will be rejected.
  - eIDAS Middleware: SAML metadata and responses no longer include line breaks in <SignatureValue> or <CipherValue>.
  - eIDAS Middleware: SAML error responses will always contain one of the allowed top level SAML status codes. Sub status codes and status message contain more specific information.
- 2.2.1
  - eIDAS Middleware: Add test mode to get SAML responses without eID interaction.
  - eIDAS Middleware: Improve SNMP integration.
  - eIDAS Middleware: The information from the SNMP integration is also shown in the admin interface.
  - eIDAS Middleware: Add the ‘method’ attribute to the SubjectConfirmation element in the SAML assertion.
  - eIDAS Middleware: Use the proxy settings also for the connection check.
  - eIDAS Middleware: The javascript check to detect if the AusweisApp2 is running is disabled for Safari browsers as they block the javascript requests to the AusweisApp2.

- eIDAS Middleware: Fix a bug introduced with 2.1.0 where the country identifier in the address would be placed in `adminunitSecondLine`
- 2.2.2
  - eIDAS Middleware: Security Patch
- 2.2.3
  - eIDAS Middleware: Change certificate chain building algorithm.
  - eIDAS Middleware: Update xmlsec.
  - eIDAS Middleware: Improve stability in trust anchor management.
- 2.2.5
  - eIDAS Middleware: Fix Log4j security issue.
- 2.2.6
  - eIDAS Middleware: Update log4j to version 2.17.1.
- 2.2.7
  - eIDAS Middleware: Update eidas-opensaml to fix a bug with the CurrentAddress and update other third party libraries for security fixes.
- 2.2.8
  - eIDAS Middleware: Security Patch

 **Attention**

Known Issue for all versions before 3.0.0: The SUN PKCS#11 security provider that is shipped with Java 8 does not support RSA-PSS signatures. In order to use an HSM and stay in line with the eIDAS cryptographic requirements, the use of EC cryptography for the SAML signature is mandatory.

- 3.0.0
  - All components: Now supporting Java 11. Support for Java 8 is discontinued.
  - eIDAS Middleware: With Java 11, the PKCS#11 provider now supports RSA-PSS signatures.
  - eIDAS Middleware: Remove support of cryptographic algorithms not following eIDAS requirements which had been kept for the Java 8 SUN PKCS#11 provider.
  - eIDAS Middleware: Remove support of older DVCA service versions, thus removing the need of different policies.
  - eIDAS Middleware: Restructure configuration.
  - eIDAS Middleware: Extend admin interface to be a configuration interface as well.
  - Configuration Migration: Add new component to facilitate migration from 2.x.
  - Remove configuration-wizard and password-generator. The functionality of these two artifacts is now implemented in the eIDAS Middleware.
- 3.0.1
  - eIDAS Middleware: Fix a bug where the wrong country code was published in the metadata.

- eIDAS Middleware: All name identifiers are published in the metadata as required by eIDAS SAML Message Format 1.2.
- eIDAS Middleware: Update DVCA server certificate in eIDAS\_Middleware\_configuration\_test.xml
- Configuration Migration: Fix a bug that migrated the wrong URL for communication with the DVCA for production systems.
- 3.1.0
  - All components: Code cleanups.
  - eIDAS Middleware: Using identglue to check the availability of the AusweisApp2.
  - eIDAS Middleware: Added a timer to renew an expired CVC with a *Request Signer Certificate* when the current CVC is expired less than two days.
    - \* Added a notification in the Admin-UI to indicate that the renewal will be tried.
  - eIDAS Middleware: Fix unsuccessful indication of a public service provider on certain conditions.
  - eIDAS Middleware: Obsolete decryption key pair for SAML has been removed in the Admin-UI.
  - eIDAS Middleware: The used holder reference will be logged if an CVC request is unsuccessful.
  - eIDAS Middleware: A sequence number is never reused for CVC requests of a service provider.
  - eIDAS Demo: Improved design and added decrypted assertion on result page.
  - eIDAS Middleware: Added support for Brainpool elliptic curves on TLS connections.
  - eIDAS Middleware: Static resources have been moved from the base path to module specific paths.
  - eIDAS Middleware: SAML redirect binding has been added.
  - eIDAS Middleware: Added support for the natural person attribute ‘Nationality’.
- 3.1.1
  - eIDAS Middleware: Update identglue and fix behaviour for mobile use.
  - eIDAS Middleware: Add the shibboleth repository to pom.xml
- 3.2.0
  - All components: Update to Java 17.
  - All components: Update to Spring Boot 3.1.
  - eIDAS Middleware: Fix use of P12 keystores for TLS keys.
  - eIDAS Middleware: Fix display of CVC availability on the status page.
  - eIDAS Middleware: Remove deprecated configuration parts in the documentation.
  - eIDAS Middleware: Only accept specified cryptographic algorithms and elliptic curve parameters.
  - eIDAS Middleware: Fix generation of sequence number after an initial CVC request.

- eIDAS Middleware: Improve form validation for initial CVC request.
- eIDAS Middleware & Demo: Update new references to the German eID client.
- 3.3.0
  - eIDAS Middleware: Database change from H2 to HSQL - breaking change
  - Database Migration Tool: Tool for migration between H2 and HSQL
  - eIDAS Middleware: Add functionality of TR-03129-1.40 - breaking change
  - eIDAS Middleware: Store Block Lists in file system instead of database, use copy in memory for fast access.
  - eIDAS Middleware: Add automatic entanglement of the TLS server certificate with the CVC
  - eIDAS Middleware: Add automatic renewal of *Request Signer Certificate*
  - eIDAS Middleware: Add automatic renewal of the TLS client certificate (at the time of publication only available in the test environment)
  - eIDAS Middleware: Remove dashboard page and set status page as landing page
  - eIDAS Middleware: Add timer history
  - eIDAS Middleware: Improved logging
  - eIDAS Middleware: Updated VM image to Debian 12 and Java 17.0.10
- 3.4.0
  - eIDAS Middleware: Add support for TLS 1.3
  - eIDAS Middleware: Change minimal key length for TLS EC certificates to 250.  
Also add a check for recommended named curves.
  - eIDAS Middleware: Change minimal key length for TLS DHE certificates to 3072.
  - eIDAS Middleware: Change SAML method for key transport to RSA-OAEP.
  - eIDAS Middleware: Remove SHA-1 cipher suites.
  - eIDAS Middleware: Add warning and log if an RSA certificate with less than 3000 bits is used for TLS.
  - eIDAS Middleware: CSR renewal will not use the current keypair.  
Also use RSA 4096 instead of ECC.
  - eIDAS Middleware: Support only ‘unspecified’ as NameIdentifier.
  - eIDAS Middleware: Change Integer objects to primitive ints in SNMPAgent: <https://github.com/Governikus/eidas-middleware/pull/31> by hduelme.
  - eIDAS Middleware: Refactor character encoding method usage: <https://github.com/Governikus/eidas-middleware/pull/32> by hduelme.
  - eIDAS Middleware: Remove checkCipherData: <https://github.com/Governikus/eidas-middleware/pull/34> by Al1c3-1337.
- 4.0.0
  - eIDAS Middleware: URL-Encode filenames for the block list files.
  - eIDAS Middleware: Add SNMP endpoint to get TLS client cert expiration date.

- eIDAS Middleware: Do not abort startup on CRL initialization errors.
  - eIDAS Middleware: The RI interface is now using version 1.40. The 1.10 version is no longer supported and the property to change the version is deleted. Breaking Change.
  - eIDAS Middleware: Added keypair certificates in XML export without private keys.
  - eIDAS Middleware: Fixed failing build caused by privileged port binding (linux) and failing tests: <https://github.com/Governikus/eidas-middleware/pull/35> by asbachb.
  - eIDAS Middleware: Added a “Delete pending certificate request” button to the Admin UI. The button is located in the CVC Info Card under the Service Provider’s details and is visible only when a pending certificate request is present.
  - eIDAS Middleware: Added a “Delete current RSC” button to the Admin UI. The button is located in the RSC Actions card under the Service Provider’s details and is visible only when a current RSC is present.
  - eIDAS Middleware: Added new column “NextRscSequenceNumber” to “TerminalPermission” table. Please make sure to create a database backup before applying this update.
  - eIDAS Middleware: Added new timer to optionally delete expired keys in the HSM
- 4.0.1
    - eIDAS Middleware: Restrict crypto algorithms to those allowed in TR-03116-4 (2025) and TR-02102-2 (2025)
    - eIDAS Middleware: Fixed error when parsing a Defect List with id-CertReplaced element.

# 10

## Glossary

### **Authorization CA**

The Authorization CA issues *Authorization Certificates* to eID Service Providers. The CA provides a SOAP Endpoint that is used by the eIDAS Middleware to request and renew *Authorization Certificates*.

### **Authorization Certificate**

The *Authorization CA* issues Authorization Certificates that enables the *eID Service Provider* to read data from the German eID Card. The Authorization Certificate contains a set of attributes that the *eID Service Provider* is allowed to access. It also contains the hash value of the SSL certificate of the *eID Service Provider* so verify that the data of the eID Card is send to the correct consumer. Because the Authorization Certificates are only a couple of days valid, the eIDAS Middleware will renew them automatically before expiration.

### **BerCA**

Part of the *German eID PKI*, see *Authorization CA*

### **Black List**

See *Block List*

### **Block List**

When a card holder reports the eID Card as lost or authorities become aware that an eID card is lost or stolen, that eID will be revoked. The Block List contains all revoked eID. The list is *eID Service Provider* specific and the eIDAS Middleware will check for every authorization request whether the eID Card is on that list. Note: the terms *Black List* and *Block List* refer to the same thing. Older versions of the specifications have coined the term *Black List* while newer versions have switched to *Block List*.

### **CHR**

Certificate Holder Reference

### **CVC**

Card Verifiable Certificate is the technical term for the *Authorization Certificate*.

### **Defect List**

In case a batch of eID cards has a defect, the certificate that was used to sign the data on this batch of cards will be added to the Defect List.

### **eID Service Provider**

In the German eID scheme the Service Provider is the instance that initially receives the data from

the eID Card. Therefore, in the eIDAS context the eID Service Provider is the eIDAS adapter of the Middleware and not the eIDAS Connector or eIDAS Service Provider.

### eIDAS Middleware

The Middleware as described in the eIDAS Technical Specification. See collaborative platform created by the European Commission<sup>20</sup>

### German eID PKI

PKI for the German eID card. See Country Verifying Certificate Authority - electronic Identity<sup>21</sup>

### Java Key Store

Java Key Store. See Wikipedia about JKS<sup>22</sup>

### Key Store

A place where keys and certificates are stored. Most often this is a file. Typical formats are [PKCS#12](#) and [Java Key Store](#)

### Master List

To verify the data from the eID Card is authenticated and has not been manipulated, digital signatures are used. The root certificates for these signatures are stored in the Master List.

### Open Virtualization Format

Open format for exchanging virtual machines.

### PKCS#12

Format for key stores. See [RFC 7292](#)<sup>23</sup>

### Request Signer Certificate

This is a long-term certificate used to sign requests for *Authorization Certificates*. It will become mandatory at some point in the future, so it is highly recommended to start using it right away.

### SNMP

Simple Network Management Protocol (SNMP) is an internet standard protocol for collecting and organizing information about managed devices on IP networks and for modifying that information to change device behavior.

### TLS

Transport Layer Security.

---

<sup>20</sup> <https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eIDAS+eID+Profile>

<sup>21</sup> [https://www.bsi.bund.de/EN/Themen/Oeffentliche-Verwaltung/Elektronische-Identitaeten/Public-Key-Infrastrukturen/CVCA/Country-Verifying-Certificate-Authority-electronic-Identity/country-verifying-certificate-authority-electronic-identity\\_node.html](https://www.bsi.bund.de/EN/Themen/Oeffentliche-Verwaltung/Elektronische-Identitaeten/Public-Key-Infrastrukturen/CVCA/Country-Verifying-Certificate-Authority-electronic-Identity/country-verifying-certificate-authority-electronic-identity_node.html)

<sup>22</sup> <https://en.wikipedia.org/wiki/Keystore>

<sup>23</sup> <https://datatracker.ietf.org/doc/html/rfc7292>

## Index

### A

Authorization CA, [39](#)  
Authorization Certificate, [39](#)

### B

BerCA, [39](#)  
Black List, [39](#)  
Block List, [39](#)

### C

CHR, [39](#)  
CVC, [39](#)

### D

Defect List, [39](#)

### E

eID Service Provider, [39](#)  
eIDAS Middleware, [40](#)

### G

German eID PKI, [40](#)

### J

Java Key Store, [40](#)

### K

Key Store, [40](#)

### M

Master List, [40](#)

### O

Open Virtualization Format, [40](#)

### P

PKCS#12, [40](#)

### R

Request Signer Certificate, [40](#)

### S

SNMP, [40](#)

### T

TLS, [40](#)