

eIDAS-Node Security Considerations

v3.1.0

eID Internal

Table of Contents

1	Introduction	5
1.1	Document structure	5
1.2	Document aims	5
1.3	Other technical reference documentation	5
2	eIDAS-Node security headers	7
2.1	Available security headers	7
2.1.1	Content Security Policy	7
2.1.2	X-XSS protection	8
2.1.3	Strict-Transport-Security	8
2.1.4	X-Frame-Options	9
2.1.5	X-Content-Type-Options	9
2.1.6	Cache control – pragma expiration	10
2.1.7	X-Robots-Tag	10
2.2	Content-Security-Policy, a front-end security header	10
2.3	X-Robots-Tag security header and exposing web crawler configuration	11
2.3.1	Method 1: Robots.txt	11
2.3.2	Method 2: X-Robots-Tag	11
2.3.3	Method 3: HTML tag<meta name="robots">	12
2.3.4	Configuration rules	12
2.3.5	Applying web crawler rules on other applications	12
3	eIDAS-Node configuration recommendations for production	13
4	Event logging	14
5	LightRequest/LightResponse interface	15
5.1	LightToken	15
5.2	Simple Protocol	15
6	Security components' interfaces	16
7	Keystores	17
8	Trust	18
8.1	Metadata used in the eIDAS-Node	18
8.1.1	Metadata exchange	18
8.1.2	Revocation of no longer entitled nodes	18
9	Cryptography	19
9.1	Clock	19
9.2	Skew time	19
9.3	Crypto dependencies	19
9.4	Security provider installation	19
9.5	Key agreement	20
9.6	Algorithms for signing	20
10	Additional considerations	21
10.1	Caches configuration	21
10.2	Ignite and TLS	21
10.3	IP Address in SAML Assertion	22
10.4	Software stack update practices	23
11	Known limitations	24
11.1	Key rollover	24
11.2	eIDAS Connector and eIDAS ProxyService should be deployed separately	24
12	Object Input Filter autoconfiguration in Ignite	25
13	References	26

Document history

Version	Date	Modification reason	Modified by
1.0	25/04/2018	Origination	DIGIT
2.1	06/07/2018	Reuse of document policy updated and version changed to match the corresponding Release.	DIGIT
2.3	20/06/2019	Adapt to be in line with the changes in CEF eIDAS-Node v2.3	DIGIT
2.4	06/12/2019	Adapt to be in line with the changes in CEF eIDAS-Node v2.4. Main changes made in sections 5.1, 7, 8.1 and 8.5. Added sections: 8.8, 8.9, and 8.10. Removed obsolete 9.3.	DIGIT
2.5	10/12/2019	Updates related to support of eIDAS Technical Specification 1.2	DIGIT
2.6	15/04/2022	Updates related to passwords security, server fingerprinting and security providers configuration	DIGIT
2.7	01/09/2023	eIDAS-Node 2.7 release	DIGIT
2.8	21/05/2024	eIDAS-Node 2.8 release	DIGIT
2.9	04/02/2025	eIDAS-Node 2.9 release	DIGIT
3.0.0	10/10/2025	eIDAS-Node 3.0.0 release	DIGIT
3.1.0	29/05/2026	eIDAS-Node 3.1.0 release	DIGIT

Disclaimer

This document is for informational purposes only and the Commission cannot be held responsible for any use which may be made of the information contained therein. References to legal acts or documentation of the European Union (EU) cannot be perceived as amending legislation in force or other EU documentation.

The document contains information of a technical nature and does not supplement or amend the terms and conditions of any procurement procedure; therefore, no compensation claim can be based on the contents of this document.

© European Union, 2023

Reuse of this document is authorised provided the source is acknowledged. The Commission's reuse policy is implemented by Commission Decision 2011/833/EU of 12 December 2011 on the reuse of Commission documents.

List of abbreviations

The following abbreviations are used within this document.

Abbreviation	Meaning
ECDH	Elliptic-Curve Diffie–Hellman
eIDAS	Electronic Identification and Signature. The Regulation (EU) N°910/2014 governs electronic identification and trust services for electronic transactions in the internal market to enable secure and seamless electronic interactions between businesses, citizens and public authorities.

Abbreviation	Meaning
CSP	Content Security Policy.
HSTS	HTTP Strict Transport Security.
IdP	Identity Provider. An institution that verifies the citizen's identity and issues an electronic ID.
MS	Member State
SAML	Security Assertion Markup Language
SP	Service Provider
XHR	XMLHttpRequest: an API in the form of an object whose methods transfer data between a web browser and a web server.
XSS	Cross-site scripting. A type of computer security vulnerability typically found in web applications.

1 Introduction

This document describes the security considerations that should be taken into account when integrating and operating the eIDAS-Node v2.5 (or higher) which implements eIDAS Technical Specifications v1.2 (<https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eIDAS+eID+Profile>).

This document focuses on the Generic parts of the eIDAS-Node, which are targeted for production. The testing tools (demo SP, demo IdP), the supplied Specific parts and the Simple Protocol, should be used for demo purposes only, and should not be deployed in your infrastructure.

Note: The URLs with ec.europa.eu domain in this document are only accessible to the experts registered with the eIDAS eID Implementation community.

1.1 Document structure

- Chapter 1 — *Introduction* this section.
- Chapter 2 — *eIDAS-Node security headers* describes the HTTP response headers that the eIDAS-Node can send in order to increase its security.
- Chapter 3 — *eIDAS-Node configuration recommendations for production* identifies the main elements to consider when installing the software in a production environment.
- Chapter 4 — *Event logging* gives the pointers to operations considerations related to handling audit logging data.
- Chapter 5 — *LightRequest/LightResponse interface* briefly describes the integration interface for background communication between eIDAS Specific and the Generic parts.
- Chapter 6 — *Security components' interfaces* identifies the interfaces of the main security components that can be implemented before installing the eIDAS-Node.
- Chapter 7 — *Keystores* identifies the keystores in eIDAS-Node.
- Chapter 8 — *Additional considerations* describes more security aspects to be considered.
- Chapter 9 — *Known limitations* identifies the main known limitations in eIDAS-Node related to security.
- Chapter 10 — References

1.2 Document aims

The aim of this document is to provide information on:

- Recommendations for setting the software security parameters, e.g. skew time;
- Operational considerations, e.g. handling logging information;
- Identification of crypto dependencies (third-party library); and
- Known limitations and opportunities for improvement.

1.3 Other technical reference documentation

We recommend that you also familiarise yourself with the following eID technical reference documents which are available on **Digital Home > eID** :

- *eIDAS-Node Installation, Configuration and Integration Quick Start Guide* describes how to quickly install a Service Provider, eIDAS-Node Connector, eIDAS-Node Proxy Service and IdP from the distributions in the release package. The distributions provide preconfigured eIDAS-Node modules for running on each of the supported application servers.
- *eIDAS-Node Installation and Configuration Guide* describes the steps involved when implementing a Basic Setup and goes on to provide detailed information required for customisation and deployment.
- *eIDAS-Node National IdP and SP Integration Guide* provides guidance by recommending one way in which eID can be integrated into your national eID infrastructure.
- *eIDAS-Node Demo Tools Installation and Configuration Guide* describes the installation and configuration settings for Demo Tools (SP and IdP) supplied with the package for basic testing.
- *eIDAS-Node and SAML* describes the W3C recommendations and how SAML XML encryption is implemented and integrated in eID. Encryption of the sensitive data carried in SAML 2.0 Requests and Assertions is discussed alongside the use of AEAD algorithms as essential building blocks.
- *eIDAS-Node Error and Event Logging* provides information on the eID implementation of error and event logging as a building block for generating an audit trail of activity on the eIDAS Network. It describes the files that are generated, the file format, the components that are monitored and the events that are recorded.
- *eIDAS-Node Error Codes* contains tables showing the error codes that could be generated by components along with a description of the error, specific behaviour and, where relevant, possible operator actions to remedy the error.

Disclaimer: The users of the eIDAS-Node sample implementation remain fully responsible for its integration with back-end systems (Service Providers and Identity Providers), testing, deployment and operation. The support and maintenance of the sample implementation, as well as any other auxiliary services, are provided by the European Commission according to the terms defined in the European Union Public License (EUPL) at https://joinup.ec.europa.eu/sites/default/files/custom-page/attachment/eupl_v1.2_en.pdf

2 eIDAS-Node security headers

This section describes the HTTP response headers that the eIDAS-Node can send in order to increase its security.

The web security model is based on the *same origin policy*. Code from `https://EidasNode:8080/EidasNode/` should only have access to `https://EidasNode:8080/EidasNode/` data, while an attacker from `https://evil.eidasNode.com` should certainly never be allowed the access. Each origin is kept isolated from the rest of the web, giving developers a safe sandbox in which to build and play. While being a good concept in theory, in practice, attackers have found different ways to subvert the system.

Cross-site scripting (XSS) attacks, for instance, bypass the same origin policy by tricking a site into delivering malicious code along with the intended content.

For more information on the security HTTP header parameters, please refer to *Additional configuration — Security* section in the *eIDAS-Node Installation and Configuration Guide*.

2.1 Available security headers

2.1.1 Content Security Policy

security.header.CSP.enabled

Content Security Policy (CSP) is a W3C standard that imposes restrictions on the origin of content. CSP prevents a wide range of attacks, including cross-site scripting (XSS) and other cross-site injections. It requires careful tuning and precise definition of the policy. If enabled, CSP has significant impact on the way the browser renders pages (e.g. inline JavaScript is disabled by default and must be explicitly allowed in a policy).

CSP can significantly reduce the risk and impact of XSS attacks in modern browsers. Its mechanism is based principally on the definition of whitelists for the sources of content (applicable to a variety of resources: scripts, fonts, stylesheets, media, images, frames).

The implementation made in the eIDAS-Node applies a set of CSP policies on all HTTP responses returned by server.

Note: Setting *security.header.CSP.enabled* also enables cache control.

```

Content-Security-Policy: default-src 'none'; object-src 'self'; style-src
'self'; img-src 'self'; font-src 'self'; connect-src 'self';script-src
'self';script-nonce
4520d9d148c4a9cbcec68e56c625943bd33551683f017b056fafad01526f5ed4;report-
uri http://localhost:8080/EidasNodeConnector/cspReportHandler;report-to
http://localhost:8080/EidasNodeConnector/cspReportHandler;frame-ancestors
'none'
X-Content-Security-Policy: default-src 'none'; object-src 'self'; style-
src 'self'; img-src 'self'; font-src 'self'; connect-src 'self';script-src
'self';script-nonce
4520d9d148c4a9cbcec68e56c625943bd33551683f017b056fafad01526f5ed4;report-
uri http://localhost:8080/EidasNodeConnector/cspReportHandler; report-to
http://localhost:8080/EidasNodeConnector/cspReportHandler;frame-ancestors
'none'
X-WebKit-CSP: default-src 'none'; object-src 'self'; style-src 'self';
img-src 'self'; font-src 'self'; connect-src 'self';script-src
'self';script-nonce
4520d9d148c4a9cbcec68e56c625943bd33551683f017b056fafad01526f5ed4;report-
uri http://localhost:8080/EidasNodeConnector/cspReportHandler;report-to
http://localhost:8080/EidasNodeConnector/cspReportHandler;frame-ancestors
'none'
x-xss-protection: 1; mode=block
x-content-type-options: nosniff
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=600000; includeSubdomains
Cache-Control: no-cache, no-store, max-age=0, must-revalidate, private
Pragma: no-cache
Expires: 0
X-Robots-Tag: noindex, nofollow
X-Powered-By:
Server:

```

Figure 1: Sample CSP header in an HTTP response header of the eIDAS-Node

When *security.header.CSP.enabled* is set to true, which is the default value, the headers *Content-Security-Policy*, *X-Content-Security-Policy* and *X-WebKit-CSP* will be included, if false those headers will not be included.

For backward compatibility with older browser versions and CSP implementations, the headers *X-Content-Security-Policy* and *X-WebKit-CSP* are also added.

An action *cspReportHandler* has been defined for logging all the CSP violations.

If it is necessary to report/log CSP violation, the property "security.header.CSP.report.uri" has to be set explicitly in *eidas.xml*.

2.1.2 X-XSS protection

security.header.CSP.XXssProtection.block

This header enables the XSS filter built into most recent web browsers. It is usually enabled by default, meaning that the role of this header is to re-enable the filter for this particular website if it was disabled by the user. This header is supported in IE 8+, and in Chrome. The anti-XSS filter was added in Chrome 4.

```
X-XSS-Protection:1; mode=block
```

Figure 3: Sample header in an HTTP response header of the eIDAS-Node

2.1.3 Strict-Transport-Security

security.header.HSTS.includeSubDomains

HTTP Strict-Transport-Security (HSTS) instructs the browser to prefer secure connections to the server (HTTP over SSL/TLS) over insecure ones. This reduces the impact of bugs in web applications like leaking session data through cookies and external links, and defends against man-in-the-middle attacks. HSTS also disables the ability for users to ignore SSL certificate warnings.

For using this, the transport protocol needs to be https, and the following configuration has to be added in *web.xml*.

Note that the *web.xml* configuration of secure and http-only cookies has only been introduced with Servlet 3.0. For older platforms, this can be configured through proprietary mechanisms, such as application server specific web descriptors.

```
<cookie-config>
  <secure>true</secure>
  <http-only>true</http-only>
</cookie-config>
```

Figure 4: Configuring HSTS in web.xml

```
Strict-Transport-Security: max-age=600000; includeSubdomains
```

Figure 5: Sample HTTP response header of the eIDAS-Node

2.1.4 X-Frame-Options

security.header.XFrameOptions.sameOrigin

These options prevent the application from rendering in a frame or iframe, which in turns protects against key logging, clickjacking and similar attacks. Values: *deny* - no rendering within a frame, *sameorigin* - no rendering if origin mismatch, *allow-from: DOMAIN* - allow rendering if framed by frame loaded from DOMAIN.

This option will limit the possibilities to frame the eIDAS-Node. The header value used by the eIDAS-Node is *sameorigin*. Therefore only applications hosted on the same origin will be allowed to frame the eIDAS node.

```
X-Frame-Options: SAMEORIGIN
```

Figure 6: X-Frame-Options — Sample header in an HTTP eIDAS-Node response header

2.1.5 X-Content-Type-Options

security.header.XContentTypeOptions.noSniff

The only defined value, '*nosniff*', prevents Internet Explorer and Google Chrome from 'MIME-sniffing', that is, from trying to infer the actual MIME type of a response (which might be different from the declared content type) by inspecting its bytes. This also applies to Google Chrome, when downloading extensions. This header reduces the exposure to drive-by download attacks and sites serving user uploaded content that, by clever naming, could be treated as executable or dynamic HTML files.

```
X-Content-Type-Options: nosniff
```

Figure 7: X-Content-Type-Options — Sample header in an HTTP eIDAS-Node response header

2.1.6 Cache control – pragma expiration

Browsers can store information for purposes of caching and history. Caching is used to improve performance, so that previously displayed information does not need to be downloaded again. History mechanisms are used for user convenience, so the user can see exactly what they saw at the time when the resource was retrieved. If sensitive information is displayed to the user (e.g. their address, credit card details, Social Security Number, or username), then this information could be stored for purposes of caching or history, and therefore retrievable through examining the browser's cache or by simply pressing the browser's **Back** button.

The cache control is enabled when *security.header.CSP.enabled* parameter is set.

The following is an example of a good header.

```
Cache-Control: no-cache, no-store, max-age=0, must-revalidate, private
Pragma: no-cache
Expires: <past date or illegal value (e.g., 0)>
```

2.1.7 X-Robots-Tag

The X-Robots-Tag instructs search engine robots how the page should be indexed. In order not to expose the Node's components, search engines are instructed with this tag not to index any page of the Node. As such, the value for this tag is defined as '*noindex, nofollow*'. This prevents both the indexing of the pages and the following of any links on the page.

```
X-Robots-Tag: noindex, nofollow
```

2.2 Content-Security-Policy, a front-end security header

The core issue exploited by XSS attacks is the browser's inability to distinguish between scripts that are intended to be part of your application, and scripts that have been maliciously injected by a third party. Instead of making browsers trust blindly *everything* that a server delivers, CSP defines the *Content-Security-Policy* HTTP header that allows the application owner to create a whitelist of sources for trusted content, and instructs the browser to only execute or render resources from those sources.

Even if an attacker can find a hole through which to inject scripts, the origin of the script will not match the whitelist, and therefore will not be executed. In the eIDAS-Node, we specify '*self*' as the only valid source of scripts. With this policy defined, the browser will simply throw an error instead of loading scripts from any other source. If an attacker does manage to inject code into the site, he will run into an error message, rather than having the success he was expecting.

The policy applies to a wide variety of resources:

- **script-src** limits the origins from which you can load scripts;
- **connect-src** limits the origins to which you can connect (via XHR, WebSockets, and EventSource);
- **font-src** specifies the origins that can serve web fonts;
- **frame-src** lists the origins that can be embedded as frames. For example: `frame-src https://youtube.com` would enable embedding YouTube videos, but no other origins;
- **img-src** defines the origins from which images can be loaded;
- **media-src** restricts the origins allowed to deliver video and audio;
- **object-src** allows control over Flash and other plugins; and
- **style-src** is the counterpart of script-src for stylesheets.

In the eIDAS-Node, the *'self'* source is used that matches only the current origin and not its subdomains.

As a consequence, inline JavaScript and CSS are not allowed, and neither are *eval* expressions.

To give an example of the impact, the following inline JavaScript

```
<body onload="document.redirectForm.submit();">
```

will need to be replaced by a JavaScript file as shown below.

```
<script src="js/redirectOnload.js"></script>
```

This file would contain the following.

```
function submitRedirectFormAction() {
    document.getElementsByName('redirectForm')[0].submit();
}
window.addEventListener('load', submitRedirectFormAction);
```

References:

<http://www.html5rocks.com/en/tutorials/security/content-security-policy/>

https://owasp.org/www-community/controls/Content_Security_Policy

2.3 X-Robots-Tag security header and exposing web crawler configuration

Search engines utilise automated programs called web crawlers to discover new web pages by following links and forms. These crawlers may initiate SAML request flows or generate unwanted traffic on endpoints during the discovery process. For a web app to indicate that a page does not contain regular HTML content and should not be indexed by search engines, three solutions are available.

Note that: While most web crawlers will respect these directives, they are under no obligation to do so.

2.3.1 Method 1: Robots.txt

The robots.txt file functions as crawler configuration and MUST be located at the host root. Its advantage over the other methods is that it will be read before any other endpoints. This file is not included in the eIDAS Node; the eIDAS Node is generally not deployed on the domain root. Administrators are responsible for its creation and maintenance.

```
User-agent: *
Disallow: /
```

Code block 1 robots.txt - Example: Block all crawlers

2.3.2 Method 2: X-Robots-Tag

The X-Robots-Tag is a response header that functions as crawler configuration, it has been included on every endpoint of the eIDAS node since version 2.6. In order to prevent the crawlers from making requests to SAML endpoints, this response header needs to be included on the pages that contain the forms for SAML redirects. Note that the noindex directive on this resource will prevent this resource from showing up in search results. The nofollow directive

only prevents links inside the resource from being called/followed and does not deflect calls on the resource itself. Thus deflecting web crawlers on SAML endpoints is a shared responsibility between you and your colleague MS.

eIDAS Node 2.5 does not contain X-Robots-Tag and we recommend upgrading your node.

```
X-Robots-Tag: noindex, nofollow
```

Code block 2 Response Header

2.3.3 Method 3: HTML tag<meta name="robots">

The HTML meta tag is a HTML tag that **can only be used on HTML pages**. In order to prevent the crawlers from making requests to SAML endpoints, this meta tag needs to be included on the pages that contain the forms for SAML redirects. Similarly to the X-Robot-Tag, it requires for a robot to load the page or servlet endpoint. Apart from the redirect page, SAML endpoints are also defined in the SAML metadata, this tag cannot be used in metadata.

This solution will not prevent requests to the first requested node page, and has not been implemented by default

```
<meta name="robots" content="noindex, nofollow">
```

Code block 3 example html tag

2.3.4 Configuration rules

Following concepts are available as values for the X-Robots-Tag response header and <meta name="robots"> HTML tag.

- **noindex** - this directive will instruct the search engine not to show the page, file, or media in search results.
- **nofollow** - this directive will instruct the search engine crawlers not to follow the links found on the page or in a document.

2.3.5 Applying web crawler rules on other applications

Preferably we would like to keep the web crawlers away from any page used for authentication, it is better if the crawler never reaches the eIDAS node. Search engine visibility is important to guide your citizens to the correct public service application (SP), so applying strict robot rules everywhere is not a good idea.

Once a citizens has navigated towards your application, they expect to see a log-in action. Any pages that follow after clicking the log-in action we consider off limits for both web crawlers and search engine results. This way we prevent users from discovering authentication pages directly from search results and we can keep the web crawlers from navigating towards the eIDAS node.

Please also be careful about exposing node endpoints in different media: Publicly shared source code, configuration, technical documents, metadata, mdsI, xml... some of these require robots rules.

3 eIDAS-Node configuration recommendations for production

The eIDAS-Node parameters allow for more flexible settings than what is restricted by the eIDAS Technical Specifications. To be compliant with the eIDAS Technical Specifications, there are several things to consider when installing the software in a production environment. For more information on this subject please see the *eIDAS-Node compliance* section in the *eIDAS-Node Installation and Configuration Guide*.

These details are more related to the environment and the application server configuration than the configuration of the application itself.

The following check list shows the features to consider:

- Follow the settings recommended in section 7.4. *eIDAS-Node compliance* of *eIDAS-Node Installation and Configuration Guide*, such as *disallow.self.signed.certificate*, *metadata.restrict.http*, and *response.encryption.mandatory*.
- Protect your application server from fingerprinting: Consult your application server documentation for further details.
- Restrict access to admin interfaces on the production server: Administrator interfaces may be present in the application or on the application server to allow certain users to undertake privileged activities on the site, access needs to be restricted (see your server documentation).
- If the application is not intended to use frames, set the protection against framing by activating the 'X-Frame-Options' header or CSP 'frame-ancestors' directive in the *eidas.xml* file (see section 2.1.4 — *X-Frame-Options*).
- In the application *web.xml*, enable secure flag on cookies and HTTP Strict Transport Security (HSTS) header (see section 2.1.3 — *Strict-Transport-Security*).
- Set the logging detail to INFO for audit trails (see the *eIDAS-Node Error and Event Logging* guide for further information).
- Configure the different caches (e.g., anti-replay cache) for production (see the *eIDAS-Node Installation and Configuration Guide* and *eIDAS-Node National IdP and SP Integration Guide*).
- Restrict the HTTP operations of your application only to the ones needed (i.e. GET and POST).
- Apply the required protection to the setting files for keystores.
- Apply the required protection to the eIDAS-Node logs.
- Apply the required protection to Ignite and TLS (see chapter Ignite and TLS).
- Apply the required protection for the Telemetry endpoints (see Installation and Configuration guide section 2.2 Modules) such as mutual TLS authentication.

The following features should be checked in the *eIDAS Technical Specifications* documents as the specifications evolve:

- Configure the application to use the correct version of TLS;
- Use cipher suites that follow the recommendations and provide perfect forward secrecy (PFS);
- Ensure that the algorithms used for signing and encryption are configured in the whitelist and that they conform to the *eIDAS Technical Specifications*.
- Ensure that the Metadata location whitelist for your node is populated and use. This is important in order to detect if the issuer in SAML messages was manipulated.

4 Event logging

eIDAS-Node produces log files with different levels of details. The detail of the logs and their formatting are described in the *eIDAS-Node Error and Event Logging* guide.

eIDAS-Node does not protect the log files it produces therefore it is recommended that they be protected according to own security operations requirements. A list of *Operations considerations* related to handling audit logging data is described in *eIDAS-Node Error and Event Logging* guide.

Moreover, note that the eIDAS-Node logs may contain person identification data, hence these logs should be handled and protected appropriately in accordance with the European privacy regulations [Dir. 95/46/EC] and [Reg. 2016/679].

[Reg. 2016/679] REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC.

[Dir. 95/46/EC] Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data.

5 LightRequest/LightResponse interface

eIDAS-Node provides *LightRequest* and *LightResponse* interface for background communication between Specific Connector and Generic Connector on the one hand, and between Specific Proxy Service and Generic Proxy Service on the other hand. For details of this interface, on the format of *LightRequest* and *LightResponse* objects, and on implementation options, please refer to section *Integration possibilities* in the *eIDAS-Node National IdP and SP Integration Guide*.

5.1 LightToken

The communication between the Specific and Generic parts is done via Shared caches (e.g. Ignite) by means of a *LightToken*. The integrity protection of the *LightToken* uses a passphrase shared between the two communicating parties, Specific and Generic. One should take appropriate measures to protect the passphrase. It is recommended that the default passphrase be changed and use reasonably long character string composed of random characters.

5.2 Simple Protocol

The eIDAS-Node integration package comes with the implementation of a Simple Protocol to exemplify the communication between a demo SP and Specific Connector, and the communication between Specific Proxy Service and a demo IdP. The Simple Protocol must not be used in production as it does not include security features.

When eIDAS-Node is integrated in MS's infrastructure it is expected that the Simple Protocol demo be replaced with the protocol used in MS's eID, e.g. SAML, OpenIDConnect.

The *LightRequest* comprises different elements to construct the eIDAS SAML *AuthnRequest* such as *spType* and *levelOfAssurance*. The *LightRequest* also contains the *relayState* (This is a feature proper to SAML, supported in all the eIDAS specified bindings). element. The Specific part can use this element to propagate some state, that will be returned with the *LightResponse*. However, it is not recommended to use it for correlation, or to expose internal state information, especially as this field is protected only by the transport layer.

Note: The Consent page implemented in the Specific part is provided only for demo purposes, thus no special protection is integrated.

6 Security components' interfaces

eIDAS-Node software implements the various interfaces defined for the main security components. One may choose to replace the implementation of certain interfaces in order to have different solutions or approach.

The interfaces of the main security components are listed below.

- eu.eidas.auth.engine.core.SAMLEngineSignI
- eu.eidas.auth.engine.core.SAMLEngineEncryptionI
- eu.eidas.auth.engine.core.ProtocolSignerI
- eu.eidas.auth.engine.metadata.MetadataSignerI
- eu.eidas.auth.engine.core.ProtocolProcessorI
- eu.eidas.auth.engine.metadata.MetadataFetcherI
- eu.eidas.auth.engine.metadata.MetadataSignerI
- eu.eidas.auth.engine.SamlEngineClock

For further details on this subject, please refer to the *eIDAS-Node and SAML* guide.

7 Keystores

eIDAS-Node keeps the keys it uses in Java keystores.

Different keystores are defined in configuration files, per Node instance, such as:

- EncryptModule_Connector.xml
- SignModule_Connector.xml
- SignModule_Service.xml

These configuration files contain *keyPassword* and *keyStorePassword* elements. eIDAS-Node does not protect these files.

It is recommended that:

1. The default passwords be changed and be reasonably long, e.g., 14, or longer character string, composed of random lowercase and uppercase letters, numbers, and special characters.
2. The configuration files be protected according to own security operations requirements.
3. The signing metadata certificate be configured in a separated keystore of the one containing the signing message certificate.
4. The content of the keystore should be as restrictive as possible. eIDAS-Node will use any private key present in the keystore to decrypt an eIDAS-Response.

Starting eIDAS-Node version 2.6, the eIDAS-Node supports integration with SunPKCS11 provider implementations (i.e., Yubico HSM).

For further details on the configuration files, please refer to the *eIDAS-Node and SAML* guide.

Note: The keys and certificates provided with the eIDAS-Node release are for demo purpose only. They should not be used in production.

8 Trust

8.1 Metadata used in the eIDAS-Node

8.1.1 Metadata exchange

To provide an uninterrupted chain of trust to securely identify the eIDAS-Nodes, two communication relationships are defined in the eIDAS Technical Specifications: eIDAS-Node Connectors with eIDAS-Node Proxy Services, and eIDAS-Node Connectors with Middleware-Services.

Concerning the relationship eIDAS-Node Connectors and Middleware-Services, as there is a one-to-one correspondence between the two entities, the certificates can be exchanged directly between the nodes.

This section focuses on metadata exchanges between eIDAS-Node Connectors and eIDAS-Node Proxy Services via the URLs and trust anchors. Trust anchors are to be exchanged bilaterally between MS.

Support for both dynamic and static use of metadata is provided. The switch parameter *metadata.http.retrieval* is available to activate the dynamic retrieval of metadata information using HTTP/HTTPS.

If the dynamic retrieval is disabled, the application will process all the files contained in a defined folder. If the static metadata is expired, the application will try to reach the remote location. Note that, the files containing the static metadata will not be updated.

When the metadata is loaded in the application, its validity is checked. An error will be printed in the logs if the metadata has expired.

8.1.2 Revocation of no longer entitled nodes

The eIDAS Technical Specifications do not specify the way the eIDAS-Nodes, and ultimately the certificates, should be revoked. However, some pointers are provided; see the excerpt from §6.2 of *eIDASInteropArch*:

"No longer entitled (e.g. due to compromise) nodes must be revoked. This can be for example done by revoking the metadata signing certificate (which revokes all nodes whose metadata are signed with that certificate) or by using short lifetimes of metadata and not resigning them in case of revocation (see also section 6.4.1)."

The eIDAS Node is able to check the metadata signing certificates for revocation. To enable this functionality, the configuration parameter *enable.certificate.revocation.checking* must be set to true, which is its default value.

More information on this functionality can be found in *Appendix A: Certificate Revocation Validation* of document *eIDAS-Node and SAML v2.8*

9 Cryptography

9.1 Clock

The clock is obtained with the implementation of a dedicated interface `eu.eidas.auth.engine.SamlEngineClock`. The default implementation uses the system time.

9.2 Skew time

The skew time gives the eIDAS-Node Connector the possibility to set a tolerance window for validating the timestamps in the SAML Responses that are sent by the eIDAS-Node Proxy Service, based on experienced clock drifts that may occur.

However, it is recommended that the clocks be synchronised between the servers responsible for authentication, thus avoiding using the skew time (with large values, or at all).

9.3 Crypto dependencies

By default, the project is built using Java SDK version 11. Running the project on Java 11 increases the TLS capabilities, e.g. support for GCM-based TLS ciphers.

eIDAS-Node Connector and eIDAS-Node Proxy Service depend on the security/crypto libraries below:

- OpenSAML
- JCE Unlimited Strength Jurisdiction Policy
- BouncyCastleProvider (in most of the cases)

For installation details on this subject, please refer to the section *Configuring the JVM* in the *eIDAS-Node Installation and Configuration Guide*.

For the complete list of dependencies, please refer to the address below (the dependencies are listed separately for each release):

<https://ec.europa.eu/digital-building-blocks/wikis/x/DWcZAq>

9.4 Security provider installation

By default BouncyCastle provider is not installed and in most cases must be installed/configured at the JVM level. The procedure is described in the eIDAS-Node installation and configuration guide.

Concerning the SunEC security provider defined by default at `java.security` file (`security.provider.3=SunEC`), it should be moved to the end of the list of available providers. The reason for this is that the SunEC implementations for the recommended Brainpool elliptic curves in the eIDAS technical specification are not secure enough, as mentioned at e.g. <https://docs.oracle.com/en/java/javase/11/security/oracle-providers.html#GUID-091BF58C-82AB-4C9C-850F-1660824D5254> and therefore should not be used in runtime. However, it cannot be removed since it is required by Ignite with TLS enabled.

9.5 Key agreement

The support for key agreement encryption with ECDH is implemented in the eIDAS-Node using OpenSAML library.

Note: By default, the key agreement sets the values of the attributes *AlgorithmID*, *PartyUInfo*, and *PartyVInfo* in *xenc11:ConcatKDFParams* to "0000". Nevertheless, the library provides the means to use other values, if need be.

9.6 Algorithms for signing

Regarding the algorithms to be used for signing, *eIDAS Cryptographic Requirement* specifies the algorithm families RSASSA-PSS and ECDSA.

From the two algorithm families, the current eIDAS-Node supports the algorithm identifiers listed below.

RSASSA-PSS

<http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-MGF1>

<http://www.w3.org/2007/05/xmldsig-more#sha384-rsa-MGF1>

<http://www.w3.org/2007/05/xmldsig-more#sha512-rsa-MGF1>

ECDSA

<http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256>

<http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384>

<http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512>

The current eIDAS-Node 2.7 no longer accepts the algorithm identifiers listed below, as being considered less secure.

RSASSA-PKCS1-v1_5

<http://www.w3.org/2001/04/xmldsig-more#rsa-sha256>

<http://www.w3.org/2001/04/xmldsig-more#rsa-sha384>

<http://www.w3.org/2001/04/xmldsig-more#rsa-sha512>

RSA-RIPEMD160

<http://www.w3.org/2001/04/xmldsig-more#rsa-ripemd160>

10 Additional considerations

10.1 Caches configuration

eIDAS-Node uses three types of caches: messages anti-replay, messages correlation, and metadata caching. There are anti-replay and correlation caches both for the messages between eIDAS nodes as well as for the messages between Specific and Generic parts. Different implementations of Jcache are provided for these caches, which can be configured. By default, the eIDAS-Node is set up to use distributed caches with different expiration values.

This implementation is provided for correlating request and reply pairs both for *AuthenticationRequests* and *LightRequests*.

Ignite and Hazelcast caches are intended to be used in production environments. The development environment may use lighter cache implementations, which are activated by setting parameters.

In order to improve the resilience of the application, we strongly recommend using the Ignite services. However, the default setting values (e.g., expiration values) should be reviewed and adapted, if necessary, before using these caches in production. Furthermore, the default certificates should be changed.

Regarding Ignite, one should use the *expiryPolicyFactory* property to specify the amount of time that must pass before an entry is considered expired.

Regarding Hazelcast, **NEVER** deploy this in an unsecured environment (eg. facing the open internet). Hazelcast by itself has no security features, so operators need to ensure hardening and security of their environment.

For installation details on this subject, please refer to the section *Advanced configuration for production environments* and to the appendix *Ignite proposed configuration* in the *eIDAS-Node Installation, and Configuration Guide* and *eIDAS-Node National IdP and SP Integration Guide*.

Note: The REST API profile is not enabled by default in Ignite. Enabling this profile adds several dependencies which may be affected by CVE's whose impact was not investigated by the eID team.

Note: It is possible to replace Ignite with alternative solutions by implementing the provided interfaces.

10.2 Ignite and TLS

By default, TLS protection is enabled for Ignite communication in eIDAS-Node.

In demo configuration files of the Ignite: (*igniteNode.xml* and *igniteSpecificCommunication.xml* from eIDAS Connector Config repository and eIDAS Proxy Config repository) it is necessary to set the following environment variables to a hard-to-guess value and to recreate their corresponding certificates. The default values (preceded by ':') can be omitted after this step.

- IGNITE_NODE_CONNECTOR_KEY_STORE_PASSWORD
- IGNITE_NODE_CONNECTOR_TRUST_STORE_PASSWORD
- IGNITE_SPECIFIC_CONNECTOR_KEY_STORE_PASSWORD
- IGNITE_SPECIFIC_CONNECTOR_TRUST_STORE_PASSWORD
- IGNITE_NODE_PROXY_KEY_STORE_PASSWORD
- IGNITE_NODE_PROXY_TRUST_STORE_PASSWORD
- IGNITE_SPECIFIC_PROXY_KEY_STORE_PASSWORD
- IGNITE_SPECIFIC_PROXY_TRUST_STORE_PASSWORD

```

<!-- Ssl/Tls context. -->
<!--IMPORTANT: THIS IS JUST A DEMO CONFIGURATION AND DEMO KEYSTORES,
NEEDS TO BE CHANGED FOR PRODUCTION ALSO THE KEYS IN THE KEYSTORES NEED TO
BE CREATED AS WELL-->
<property name="sslContextFactory">
  <bean class="org.apache.ignite.ssl.SslContextFactory">
    <property name="keyStoreFilePath"
value="${EIDAS_CONNECTOR_CONFIG_REPOSITORY}/ignite/KeyStore/server.jks"/>
    <property name="keyStorePassword"
value="${IGNITE_NODE_CONNECTOR_KEY_STORE_PASSWORD:123456}"/>
    <property name="trustStoreFilePath"
value="${EIDAS_CONNECTOR_CONFIG_REPOSITORY}/ignite/KeyStore/trust.jks"/>
    <property name="trustStorePassword"
value="${IGNITE_NODE_CONNECTOR_TRUST_STORE_PASSWORD:123456}"/>
    <property name="protocol" value="TLSv1.2"/>
  </bean>
</property>

```

```

<!-- Ssl/Tls context. -->
<!--IMPORTANT: THIS IS JUST A DEMO CONFIGURATION AND DEMO KEYSTORES,
NEEDS TO BE CHANGED FOR PRODUCTION ALSO THE KEYS IN THE KEYSTORES NEED TO
BE CREATED AS WELL-->
<property name="sslContextFactory">
  <bean class="org.apache.ignite.ssl.SslContextFactory">
    <property name="keyStoreFilePath"
value="${EIDAS_PROXY_CONFIG_REPOSITORY}/ignite/KeyStore/server.jks"/>
    <property name="keyStorePassword"
value="${IGNITE_NODE_PROXY_KEY_STORE_PASSWORD:123456}"/>
    <property name="trustStoreFilePath"
value="${EIDAS_PROXY_CONFIG_REPOSITORY}/ignite/KeyStore/trust.jks"/>
    <property name="trustStorePassword"
value="${IGNITE_NODE_PROXY_TRUST_STORE_PASSWORD:123456}"/>
    <property name="protocol" value="TLSv1.2"/>
  </bean>
</property>

```

It is also recommended that you protect these configuration files according to your own security requirements.

Please also check/follow the TLS requirements stated at eIDAS technical specification, more exactly the ones stated at eIDAS Cryptographic Requirement for the Interoperability Framework TLS and SAML Version 1.2.

For more details, please refer to section *Enable TLSv1.2 Ignite nodes* in the *eIDAS-Node Installation and Configuration Guide*.

10.3 IP Address in SAML Assertion

By default, eIDAS-ProxyService does not add the attribute Address in the element *SubjectConfirmationData* of SAML Assertion it creates. One can enable the addition of this attribute with a dedicated configuration property:
enable.address.attribute.subject.confirmation.data

Note: When this property is enabled, the IP address is obtained from *x-forwarded-for* HTTP header or, if this is not present, from the IP address of the HTTP Request sender. Given the fact that the use of this HTTP header may not be fully trusted, enabling adding Address in SAML Assertions should be used with caution.

10.4 Software stack update practices

It is recommended that eIDAS-Node deployments follow the general best practices with regard to updates of the entire software stack surrounding the eIDAS-Node (i.e., OS, Middleware, JDK), and especially with regard to security patches.

11 Known limitations

This section lists the main known limitations in eIDAS-Node related to security. They represent a subset of the overall eIDAS-Node limitations kept up to date at the address below (the limitations are listed separately for each release).

<https://ec.europa.eu/digital-building-blocks/wikis/x/DWcZAq>

It is recommended to regularly check this list for any eventual new reported limitations.

As part of our regular security scanning activities, we run dependency checks. At the address below the eID team announces the reported vulnerabilities.

<https://ec.europa.eu/digital-building-blocks/wikis/x/CwB2Aq>

It is recommended to regularly check this list for any eventual new reported vulnerabilities and the outcome of investigation (made by the eID team) of their eventual impact.

It is also encouraged that MS run vulnerability and dependency checks on their eIDAS integration and eventually report findings to the eID team.

Note: The investigation of eventual impact of reported vulnerabilities is being done considering the way and the context the dependencies and the related code are used in eIDAS-Node, as released. Subsequent changes in usage, context, or behaviour made to the eIDAS-Node releases by eIDAS-Node implementers should to be assessed considering any related reported vulnerabilities.

11.1 Key rollover

In order to facilitate the rollover for signing and encryption certificates, a node may have metadata with more than one KeyDescriptor for the same usage. During rollover, both keys can be used. It is advised to be in rollover state for at least one metadata validity period.

Note that rollover will not work when communicating with an eIDAS Node prior to version 2.9 For a more detailed description of rollover, please refer to the eIDAS-Node and SAML document, section 8.2: Certificate Rollover

11.2 eIDAS Connector and eIDAS ProxyService should be deployed separately

As of version 2.7, the eIDAS Connector and eIDAS ProxyService are built as separate .war files.

In a production environment, it is recommended to run Connector and ProxyService on different application servers (ideally on separate hosts). This approach limits the ability to interfere with the other module in a malicious way.

(Example: recommendation found in Security Considerations of Apache Tomcat)

12 Object Input Filter autoconfiguration in Ignite

Apache Ignite 2.17.0 enforces stricter deserialization filtering via `java.io.ObjectInputFilter` to mitigate deserialization vulnerabilities and enhance security. This filter is set globally per JVM and can only be configured once. Attempting to reconfigure it across multiple Ignite-enabled applications in the same JVM (not recommended) can result in startup errors. To support this scenario safely, a default allowlist file (`ignite-whitelist.txt`) is included in the classpath and automatically registered at startup by the `IgniteUtils.configureIgniteWhitelistDefault()` method. This method sets both `IGNITE_MARSHALLER_WHITELIST` and `IGNITE_ENABLE_OBJECT_INPUT_FILTER_AUTOCONFIGURATION` system properties to ensure secure deserialization using a predefined list of allowed classes.

All required eIDAS classes for Ignite deserialization have been included in the default whitelist. However, running a second Ignite-enabled application in the same JVM is only possible if it requires the same set of whitelisted classes. Otherwise, class mismatches may result in Ignite 2.17+ startup errors, to prevent runtime tampering. For better isolation and security, it is strongly recommended to run each Ignite-enabled module in its own JVM (e.g., using containerized deployments), avoiding shared filter configurations altogether.

The eIDAS-Node relies on Jakarta XML Binding (JAXB) to serialize data before it is stored or retrieved from the caching solution. Operators must ensure access to the caching solution is restricted to the private network.

13 References

- *eIDASCryptoReq*: eIDAS - Cryptographic Requirements for the Interoperability Framework
- *eIDASInteropArch*: eIDAS - Interoperability Architecture