eIDAS-Node Security Considerations

Version 2.5

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	11/12/2019	Updates related to support of eIDAS Technical Specification 1.2	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.

Table of contents

Table of figures

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.
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 CEF eIDAS-Node v2.5 (or higher) which implements eIDAS Technical Specifications v1.2 [1] and provides backwards compatibility with CEF eIDAS-Node v.2.x - 2.4 by supporting eIDAS Technical Specifications v1.1

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.

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

[1] eIDAS Technical Specifications v1.2 https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eIDAS+eID+Profile

- 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 **CEF Digital Home > eID > Services > eIDAS Node Integration Package**

- *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.
- *CEF eID 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.

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'; xhr-src 'self'; font-src 'self'; connect-src
'self';script-src 'self';script-nonce
bf9727e96e349e5d40bb439bc46ed9bdb9f7e342;report-uri http://vs-cis-
k2:8081/EidasNode/cspReportHandler

X-Content-Security-Policy: default-src 'none'; object-src 'self'; style-src 'self';
img-src 'self'; font-src 'self'; xhr-src 'self'; font-src 'self'; connect-src
'self';script-src 'self';script-nonce
bf9727e96e349e5d40bb439bc46ed9bdb9f7e342;report-uri http://vs-cis-
k2:8081/EidasNode/cspReportHandler

X-WebKit-CSP: default-src 'none'; object-src 'self'; style-src 'self'; img-src
'self'; font-src 'self'; xhr-src 'self'; font-src 'self'; connect-src
'self';script-src 'self';script-nonce
```

```
bf9727e96e349e5d40bb439bc46ed9bdb9f7e342;report-uri http://vs-cis-
k2:8081/EidasNode/cspReportHandler

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
```

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

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 the browser is too old or if for another reason the CSP is not applied, a default error message will appear in the header of the application. This behaviour is disabled by default, because it is triggered by an enforced CSP violation, which is always displayed in server logs by the CSP report servlet.

To enable this feature, `security.header.CSP.fallbackCheckMode` must be configured in `eidas.xml`.

In order to limit the risks of being exposed to threats known as "host header poisoning", one should fill in the parameter `security.header.CSP.report.uri` . This value should indicate the domain (URL prefix) for the URL to be used with the `report-uri` header.

2.1.2. Mozilla original directive in CSP: xhr-src

`security.header.CSP.includeMozillaDirectives`

The original Firefox implementation of CSP used the '`xhr-src`' directive to restrict the origins to which an `XMLHttpRequest` object can connect. In the 1.0 specification, '`xhr-src`' was replaced by '`connect-src`' – which, in addition to XHR, also restricts where `EventSource` and `WebSocket` objects can connect.

```
Content-Security-Policy: default-src 'none'; object-src 'self'; style-src 'self';
img-src 'self'; font-src 'self'; xhr-src 'self'; font-src 'self'; connect-src
'self';script-src 'self';script-nonce
bf9727e96e349e5d40bb439bc46ed9bdb9f7e342;report-uri http://vs-cis-
k2:8081/EidasNode/cspReportHandler
```

Figure 2: Sample CSP header in an HTTP response header of the eIDAS-Node

2.1.3. 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

2.1.4. 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`[2]:

```
<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

2.1.5. 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

[2] 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.

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

2.1.6. 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

2.1.7. 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)>
```

2.2. Development guidelines

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://www.owasp.org/index.php/Content_Security_Policy

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 *7.5. 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.
- 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 2.1.5 — *X-Frame-Options*).
- In the application `web.xml`, enable secure flag on cookies and HTTP Strict Transport Security (HSTS) header (see section 2.1.4 — *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.

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

CEF 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, Hazelcast) 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 CEF 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 CEF 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`[3] 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.

[3] This is a feature proper to SAML, supported in all the eIDAS specified bindings.

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
EncryptModule_Service.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.

Currently, eIDAS-Node does not provide an interface to support integration with HSM (i.e., PKCS11 standard).

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. Additional considerations

8.1. 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 `AuthenticationRequest`s and `LightRequest`s.

Ignite 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 three types of caches; either with Ignite or Hazelcast 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 configuration, one should pay attention to `time-to-live-seconds` and `eviction-policy`. These parameters limit the exposure for a takeover of the flow by a third party (even though other security countermeasures and mechanisms applied with eIDAS should prevent it from happening).

For installation details on this subject, please refer to the section *Advanced configuration for production environments* and to the appendixes *Ignite proposed configuration* and *Hazelcast 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 CEF eID team.

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

8.2. Metadata used in the eIDAS-Node

8.2.1. 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.

8.2.2. 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)."*

8.3. Clock

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

8.4. 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).

8.5. Crypto dependencies

By default, the project is built using Java SDK version 1.8. Running the project on Java 1.8 increases the TLS capabilities, e.g. support for GCM-based TLS ciphers.

In order to avoid a possible XML External Entity attack (XXE), the OWASP guidelines advise to use Java 8 update 20 or above. It is recommended to install the latest JDK security patches as soon as those become available.

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

- OpenSAML
- SwedenConnect opensaml-security-ext
- JCE Unlimited Strength Jurisdiction Policy
- BouncyCastleProvider (in case of IBM SDK Java)

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/cefdigital/wiki/x/DWcZAg

8.6. Ignite and TLS

By default, TLS protection is enabled for Ignite communication in eIDAS-Node. For more details, please refer to section *Enable TLSv1.2 Ignite nodes* in the *eIDAS-Node Installation and Configuration Guide*.

8.7. Hazelcast and TLS

The TLS protection of the Hazelcast communication is not enabled. Therefore it is recommended that the components using Hazelcast be located within a protected infrastructure.

8.8. 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 carefully used.

8.9. Security provider reinstallation

When multiple applications using BouncyCastle are deployed on the same application server, ClassCastException can be raised when redeploying one of the applications. This can occur when a JCE is loaded at global JVM level.

To disable the reinstallation of JCE providers, a system property was defined: block.security.provider.reinstall.

8.10. Key agreement

The support for key agreement encryption with ECDH added in eIDAS-Node is based on OpenSAML extension se.swedenconnect.opensaml:opensaml-security-ext [4]. To date, key agreement with ECDH is not supported by the main stream library OpenSAML used in CEF implementation.

Note: By default, the key agreement in opensaml-security-ext 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.

8.11. 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.

[4] https://github.com/swedenconnect/opensaml-security-ext

9. 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/cefdigital/wiki/x/DWcZAg

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 CEF eID team announces the reported vulnerabilities.

https://ec.europa.eu/cefdigital/wiki/x/CwB2Ag

It is recommended to regularly check this list for any eventual new reported vulnerabilities and the outcome of investigation (made by CEF 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 CEF 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.

9.1. Algorithms for signing

Regarding the algorithms to be used for signing, *eIDASCryptoReq* specifies the algorithm families RSASSA-PSS and ECDSA. However, it does not specify[5] the actual algorithm identifiers to be supported in eIDAS.

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

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 also supports the algorithm identifiers listed below, which became obsolete.

[5] This was addressed in the new version of eIDAS Technical Specifications v1.2, released in Sept 2019.

These algorithms are not recommended to be used in eIDAS.

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

9.2. Key rollover

Currently, metadata is loaded into the application when the eIDAS-Node is started. Thus, updates in metadata and keys will not be used by the application until the application is restarted.

In order to facilitate the rollover for metadata signing key, a node may have metadata with more than one KeyDescriptor for the same usage. However, the eIDAS-Node reads only one KeyDescriptor for the same usage.

9.3. eIDAS Connector and eIDAS ProxyService should be separate components

Currently, eIDAS-Node implements both the eIDAS-Connector and the eIDAS-ProxyService in the same application. The functionality (role) of one or the other is activated via configuration parameters.

The following would represent several advantages if the two would be split in separate components / applications:

- Neat distinction in the roles and configuration parameters,
- Less risk for misconfiguration,
- Unnecessary code is not deployed if not used.

10. References

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