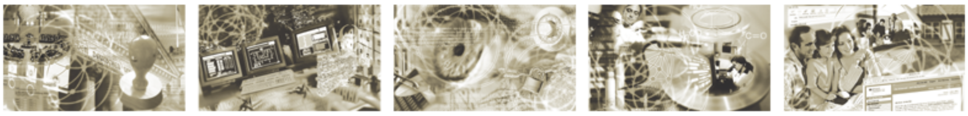
Bundesamt
für Sicherheit in der
Informationstechnik



BSI Technical Guideline 03125
Preservation of Evidence of Cryptographically Signed Documents

Appendix to Annex TR-ESOR-F:

Annex TR-ESOR-AeSV: AIP-eIDAS-Signature-Validator-Tool

Designation	AIP-eIDAS-Signature-Validator-Tool (AeSV)
Abbreviation	BSI TR-ESOR-AeSV
Version	1.2.2 (on base of the eIDAS-Regulation and the ETSI Preservation Standards)
Date	25.02.2021

Federal Office for Information Security
Post Box 20 03 63
53133 Bonn
Phone: +49 228 99 9582-0
E-Mail: tresor@bsi.bund.de
Internet: https://www.bsi.bund.de

Content

Table of Figures

Table List

1. Introduction

1.1 Overview

This document contains an overview of a validation component[1] for XAIP objects pursuant to ([TR-ESOR-F], clause 3.1) or LXAIP objects pursuant to (**[TR-ESOR-F]**, clause 3.2)[2].
(L)XAIP means XAIP or LXAIP, including electronic signatures or seals or time-stamps.
The AIP-eIDAS-SignatureValidator-Tool (short: AIP-validator) contains functionality for the structural and syntactical validation of (L)XAIP preservation objects. It offers additionally the possibility of using an external digital signature verification service for the purpose of validating electronic signatures and timestamps.

1.2 Legal and other information

The AIP-eIDAS-Signature-Validator-Tool (AeSV) itself as well as this documentation are provided under the Apache License Version 2.0, which is enclosed to the product:

1.3 Scope of the document

The document covers the following aspects:

[1] Further named as AIP-validator.

[2] LXAIP: a variant of XAIP, where there may be a reference to externally stored data objects in the ECM/Long-Term Storage

- Installation and configuration
- Parameterization and usage
- Integration of signature verification components
- Major design decisions
- Known limitations

This documentation applies to the code located on https://github.com/de-bund-bsi-tr-esor/tr-esor-AIP-eIDAS-SigValidator marked with tag 1.0.8.

2. Introduction

The Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik, BSI) is the responsible German authority for secure information technology and especially for preservation on base of digital signature techniques.

The BSI has developed a technical guideline for the long-term preservation evidence of cryptographically signed, signed or timestamped documents (TR-03125/TR-ESOR). A key aspect of this technical guideline is the preservation of electronic documents in XML-based structures named XAIP-containers.
The core feature of XAIP-containers is the ability of storing self-contained documents, meta-data and credentials containing electronic signatures.

The creation of XAIP-containers is a typical task in the context of preservation solutions. The technical guideline TR-03125 TR-ESOR introduces XML-adapters. The client application or the XML-adapters are responsible for the fulfillment of this task. Producing such containers raises the demand for tools that are able to verify the structural and syntactical correctness as well as electronic signatures, seals and timestamps, which may be part of the container.

The AIP-eIDAS-Signatur-Validator addresses this demand. It provides the following functionality:

- Syntactical analysis: verification if well-formedness of XAIP objects
- Structural analysis: verification of required and optional elements in the XAIP object
- Signature, seal and timestamp validation: Validation of electronic signatures, seals and timestamps by using a verification service conformant to the signature verification functionality defined by the technical guideline TR-03112
- Creation of verification protocol conformant to TR-ESOR 03125, Annex TR-ESOR-VR integrating signature validation reports conforming to OASIS DSS v1.0 profile for comprehensive multi-signature verification reports version 1.0

The main building blocks of the AIP-eIDAS-Signature-Validator are the following specifications:

- TR-03125, Anlage TR-ESOR-F: Formate Version 1.2.2 [TR-ESOR-F]
- TR-03125, Anlage TR-ESOR-VR: Verification Reports for Selected Data Structures, Version 1.2.1 [TR-ESOR-VR]

The validator does not depend on any preservation solution or any other third-party system with exception of the signature verification service. Figure 1 depicts the general design overview. The following interfaces and components are part of the AIP-validator:

Component	**Purpose**
CLI	Interface for accessing the functionality from command line
SOAP	Interface for accessing the functionality using SOAP

Component	Purpose
AIP-eIDAS-Signature-Validator-Tool (short: AIP-Validator)	The AIP-validator component comprising modules for syntax validation, signature identification, signature verification and protocol generation.
Dispatcher	The dispatcher module is responsible for configuration evaluation as well as configuring other validator modules. It manages the calls to the modules with validation functionality.
SyntaxValidator	A module responsible for syntax validation. Checks the XAIP container presented to the validator for syntactical correctness and provides validation results to the Dispatcher.
SignatureFinder	A module responsible for identification of electronic signatures and timestamps in the XAIP container. Provides information about electronic signatures and timestamps to the SignatureVerifier.
SignatureVerifier	This component is responsible for calling the external Signature Verification Service with the signatures and timestamps identified during the XAIP inspection.
ProtocolAssembler	The ProtocolAssembler is responsible for the generation of the protocol stating the results of XAIP inspection and signature validation.
Signature Verification Service	The external Service for verification of electronic signatures and timestamps that are part of the XAIP-container. The component provides the interface named 'Signature Verification' that is compliant to the specification of 'VerifyRequest' in TR-03112. All signature verification reports including those for electronic timestamps embedded into the validation report are generated by the external signature validation service. The AIP-validators capabilities are limited to the identification of signatures and timestamps and requesting the validation by the external verification component.

Table 1 Design Elements of the AIP-Validator

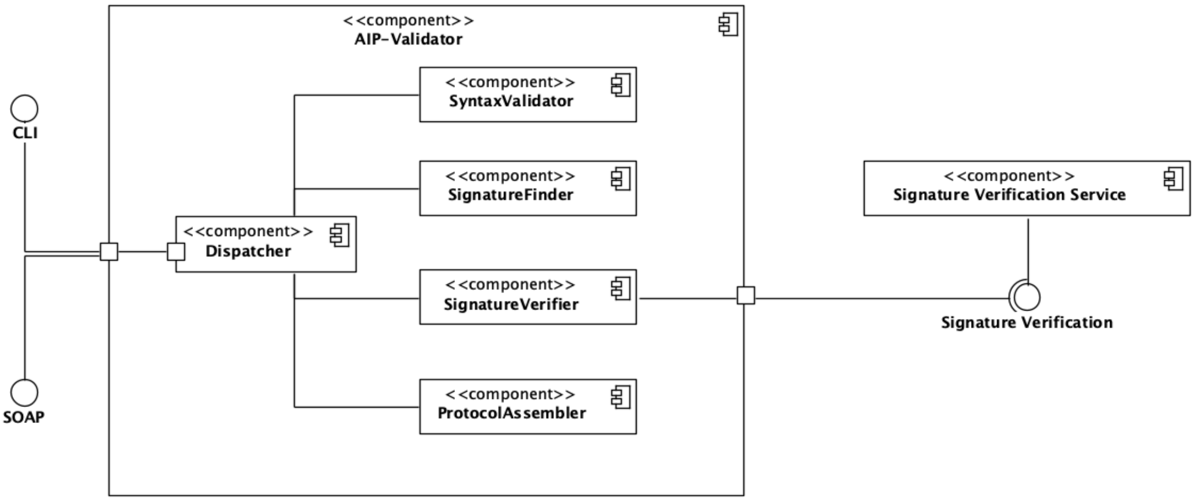


Figure 1 General Design Overview

The source code of the AIP-validator is located on GitHub under de-bund-tr-esor/tr-esor-AIP-eIDAS-SigValidator.

3. Installation and Configuration

3.1 Prerequisites

General information

The following documentation is based on the assumption that the system used has a current patch status and that applications and libraries are kept up to date. It must be ensured that these requirements are met in order to ensure the functionality of the components to be installed.

Java 11

First you should install a current Java SDK. To build and run the AIP-Validator, at least Java 11 from version 11.0.10 is required. You can check the currently used version on the command line using the command **java** --version.

Git-Client

If the code is to be retrieved directly from the repository, the GIT command line tools and a GIT client - for example GitHub Client or Source Tree or another client of your choice - are required. You may need to have a current GitHub account.

You can find further information at this point: https://github.com/git-guides/install-git.

A list of graphical git clients can be found here: https://git-scm.com/downloads/guis.

Maven

In order to build the AIP-Validator, Maven is required. Maven and its documentation can be downloaded here: http://maven.apache.org/index.html.

3.2 Repository access

Depending on the selected access method, the repository can be cloned. The following call shows an example with stored SSH keys. Alternatively, the code archive can be downloaded as a ZIP file. In this case, it is not required to access the repository via a git client.

```
git clone git@github.com:de-bund-bsi-tr-esor/tr-esor-AIP-eIDAS-SigValidator.git
```

More information about cloning git repositories can be found here: https://github.com/git-guides/git-clone.

3.3 Maven Build

For building the project follow these steps:

- Open a terminal and change the current directory to the project root directory
- Call the Maven build command **mvn clean** package

The following lines demonstrate the procedure when the project root is named xaipvalidator[3].

```
cd xaipvalidator

mvn clean package
```

The readme file of the AIP-Validator project on GitHub contains a complete example of how to build the tool including the JavaDoc documentation under https://github.com/de-bund-bsi-tr-esor/tr-esor-AIP-eIDAS-SigValidator/blob/master/README.md.

The results of the build procedure will be located `xaip-validator-cli/target/`.

3.4 Creating JavaDoc API documentation

The Javadoc API documentation can be built calling:

```
mvn javadoc:aggregate -Ddoclint=none
```

on the root project. The javadoc will be available under tr-esor-AIP-eIDAS-SigValidator/target/site/apidocs/index.html which should be opened using an internet browser.

3.5 CLI Options

-M=`<moduleConfig>`

Passing a single module configuration property to the validator. The property and requirements for any module configuration properties are defined by the implementing module.

Example:

```
-Mverifier.wsdlUrl=http://localhost:8080/s4?wsdl
```

This example is using the property config verifier.wsdlUrl which is a required config property of the DefaultVerifierModule.

Instead of passing multiple module properties as an argument, a config file which contains the module properties can be passed instead. For more information see
`–c, --config <file>`

[3] The standard root directory of the project is identical to name of the project on GitHub (tr-esor-AIP-eIDAS-SigValidator).

-c, --config <file>

Passing a configuration file in form of a property file to the validator. This file can contain any module configuration property. The configuration file is being passed to every module so they can scan the content and retrieve any configuration properties they define.

The handling is just the same as passing all configuration properties via a separate command line argument.

This argument can also be used together with single command line module property arguments.

Example:

```
-c config.properties
```

Content of `config.properties:`

```
validator.schemaDir=/tmp/xaip/definition
verifier.wsdlUrl=http://localhost:8080/s4?wsdl
```

-i, --in, --input <file>

Passing a <file> as a source for the xaip validation. Omitting this argument will take the standard <inputStream> for the validation.

Example:

```
-i /tmp/sample.xaip
```

-o, --out, --output <file>

Defining a definition for the validation result. Omitting this argument will write the result to the standard <OutputStream> instead.

Example:

```
-o /tmp/report.xml
```

-v, --verify

This flag enables the signature verification which is being executed by the [SignatureVerifierModule]. Omitting this flag will only execute the syntax validation.

`-d, --debug`

Flag to enable debug output for a better analysis of the validator behaviour. This output can contain stack traces or other kinds of errors even when everything works fine.

`-l, --log <file>`

Since this tool is not only creating a report but also log output this argument can be used to separate the log output of the validator into a dedicated document.

Example:

```
-l validator.log
```

`-h, --help`

Displays information for using the validator.

Important Notes

- When using the command line version of the AIP-Validator, the location of the schema files by using the parameter `-Mvalidator.schemaDir` is required unless a custom syntax validation module is used. Otherwise, the validator will not start.
- The argument of parameter `-o` must specify a directory. Currently the validator cannot write a plain file located in the same directory.
- When using parameter `-v` the URL of the signature verification service must be specified unless you are using a custom verification module. The signature verification service must be compliant to the TR-03112 / OASIS interface definition (using `VerifyRequest`).

3.6 Server Options

`-M=``<moduleConfig>`

Passing a single module configuration property to the validator. The property and requirements for any module configuration properties are defined by the implementing module.

Example:

```
-Mverifier.wsdlUrl=http://localhost:8080/s4?wsdl
```

This example is using the property config `verifier.wsdlUrl` which is a required config property of the `DefaultVerifierModule`. Instead of passing multiple module properties as an argument, a config file which contains the module properties can be passed instead.

For more information see `-c, --config <file>`

`-c, --config` `<file>`

Passing a configuration file in form of a property file to the validator. This file can contain any module configuration property. The configuration file is being passed to every module so they can scan the content and retrieve any configuration properties they define.

The handling is just the same as passing all configuration properties via a separate command line argument. This argument can also be used together with single command line module property arguments.

Example:

```
-c config.properties
```

Content of `config.properties`:

```
validator.schemaDir=/tmp/xaip/definition
verifier.wsdlUrl=http://localhost:8080/s4?wsdl
```

`-p, --port` `<port>`

Port the server should be published to, 8080 by default

`-P, --protocol` `<protocol>`

Protocol to be used, `http` by default

`-H, --host` `<hostname>`

Hostname the server is published to

`--path` `<path>`

Custom path the service should be used, `/xaip-validate` by default

`-d, --debug`

Flag to enable debug output for a better analysis of the validator behavior. This output may contain stack traces or other types of error messages even when everything works fine.

-l, --log <file>

Since this tool is not only creating a report but also log output this argument can be used to separate the log output of the validator into a dedicated document

Example: -l validator.log

-h, --help

Displays information for using the validator.

3.7 Available parameters of the validator modules

Without any additional configuration, the AIP-validator is using the default module implementations. Every module implementation can specify their own configuration which can be passed from outside using the param -M or −config. They can also be specified as a requirement to use the module.

All configuration options labelled with an asterisk (*) are mandatory configuration properties which have to be provided as a minimum requirement to use the module.

Parameter name	**Example**	**Description**
*validator.schemaDir	/tmp/xaip-schema	Path to the schema directory containing the xaip schema files

Table 2 Parameters of the Default SyntaxValidator Module

Parameter name	**Example**	**Description**
*validator.wsdlUrl	https://host:port/VerificationService/S4?wsdl	URL to the WSDL of the VerificationService which should be used

Table 3 Parameters of the Default Verifier Module

4. Usage

4.1 Command Line Interface (CLI) usage

```
java
     -jar xaip-validator-cli/target/xaip-validator-cli.jar
     -i ~/example.xaip
     -Mvalidator.schemaDir=<schema-directory>
```

4.2 SOAP-Server mode usage

4.2.1 MacOS and Linux

```
java
     -cp "xaip-validator-soap/target/xaip-validator-soap-1.0.6-
     1.jar:target/dependency/*"
     de.bund.bsi.tresor.xaip.validator.soap.Server
     -Mvalidator.schemaDir=<schema-director>
     -Mverifier.wsdlUrl=<verification-service-url>
```

Example of the verification service URL:

```
"https://host:port/VerificationService/S4?wsdl"
```

4.2.2 Windows

```
java
     -cp "xaip-validator-soap/target/xaip-validator-soap-1.0.6-
     1.jar;target/dependency/*"
     de.bund.bsi.tresor.xaip.validator.soap.Server
     -Mvalidator.schemaDir=<schema-directory>
     -Mverifier.wsdlUrl=<verification-service-url>
```

Example of the verification service URL:

```
"https://host:port/VerificationService/S4?wsdl"
```

5. Integration of Time-Stamp Verification Components

By default, the AIP-Validator utilizes a signature verification component that is compliant to the eCard-Framework (TR-03112) by implementing the web service method VerifyRequest. This web service method is part of the web service definition file ‘eCard.wsdl’, which is part of the web service and schema definitions provided together with the eCard-Framework.

```
<wsdl:operation name="VerifyRequest">
<soap:operation soapAction="http://www.bsi.bund.de/ecard/api/1.1#VerifyRequest" />
        <wsdl:input>
                <soap:body use="literal" />
                        </wsdl:input>
                        <wsdl:output>
                <soap:body use="literal" />
        </wsdl:output>
</wsdl:operation>
```

Implementations compliant to the specification of the eCard-Framework [TR-03112] can be utilized for signature verification by setting the service url as a configuration parameter of the XAIP-validator.

5.1 Recommendation

For configuration of the signature verification webservice to be used; the parameter `verifier.wsdlUrl` must be set. For adjustment of the utilized signature verification implementation the use of an eCard-Framework compliant verification service is recommended.

5.2 Extending the implementation

In case that the use of an eCard-Framework compliant service is not possible; the module implementation must be extended. This section explains a possible approach.

Setting up a verification module

A custom signature verification module should be implemented for supporting a custom signature verification service. The signature verification module must implement the interface `SignatureVerifier` from package `de.bund.bsi.tresor.xaip.validator.api.boundary`.

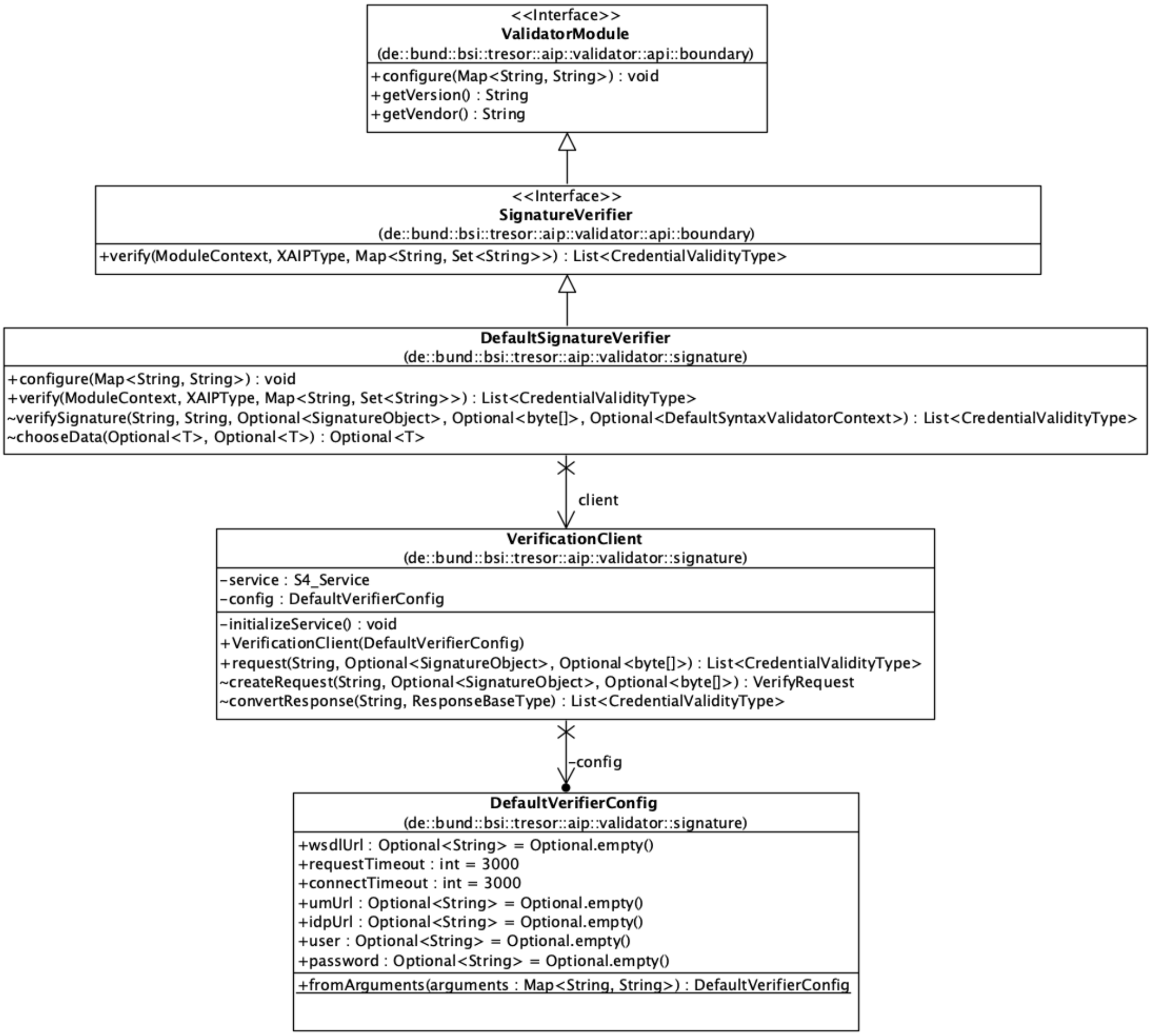

<<Interface>>
ValidatorModule
(de::bund::bsi::tresor::aip::validator::api::boundary)
+configure(Map<String, String>) : void
+getVersion() : String
+getVendor() : String
<<Interface>>
SignatureVerifier
(de::bund::bsi::tresor::aip::validator::api::boundary)
+verify(ModuleContext, XAIPType, Map<String, Set<String>>) : List<CredentialValidityType>
DefaultSignatureVerifier
(de::bund::bsi::tresor::aip::validator::signature)
+configure(Map<String, String>) : void
+verify(ModuleContext, XAIPType, Map<String, Set<String>>) : List<CredentialValidityType>
~verifySignature(String, String, Optional<SignatureObject>, Optional<byte[]>, Optional<DefaultSyntaxValidatorContext>) : List<CredentialValidityType>
~chooseData(Optional<T>, Optional<T>) : Optional<T>
client
VerificationClient
(de::bund::bsi::tresor::aip::validator::signature)
-service : S4_Service
-config : DefaultVerifierConfig
-initializeService() : void
+VerificationClient(DefaultVerifierConfig)
+request(String, Optional<SignatureObject>, Optional<byte[]>) : List<CredentialValidityType>
~createRequest(String, Optional<SignatureObject>, Optional<byte[]>) : VerifyRequest
~convertResponse(String, ResponseBaseType) : List<CredentialValidityType>
-config
DefaultVerifierConfig
(de::bund::bsi::tresor::aip::validator::signature)
+wsdlUrl : Optional<String> = Optional.empty()
+requestTimeout : int = 3000
+connectTimeout : int = 3000
+umUrl : Optional<String> = Optional.empty()
+idpUrl : Optional<String> = Optional.empty()
+user : Optional<String> = Optional.empty()
+password : Optional<String> = Optional.empty()
+fromArguments(arguments : Map<String, String>) : DefaultVerifierConfig

Figure 2 Class hierarchy and dependencies of DefaultSignatureVerifier

5.2.1 Required Methods

The following methods need to be implemented by a signature verification module.

```
public String getVersion()
```

Provides a printable string with the version of the module, e.g., 1.0.0.

```
public String getVendor()
```

Provides a printable string with the name of the module vendor, e.g., “BSI”.

```
default public void configure(
     Map<String, String> properties )
```

Provides a function that receives configuration parameters and performs internal configuration tasks.

```
public List<CredentialValidityType> verify(
      ModuleContext context,
      XAIPType xaip,
      Map<String, Set<String>> credIdsByDataId );
```

This method is called automatically by the implementation of the XAIP-validator.

`ModuleContext context`

> The `ModuleContext` instance is an object that can be used to transfer additional information that needs to be exchanged between different modules.

`XAIPType xaip`

> The xaip-object is an object representation of the XAIP-container processed by the XAIP-validator.

`Map<String, Set<String>> credIdsByDataId`

> This map contains the detached signatures and timestamps found by the XAIP-validator during XAIP-container analysis.

Figure 2 shows a dependency to class `VerificationClient`. This client is responsible for sending the verification request to the signature verification web service. A possible strategy for the integration of a different signature validation component is the modification of this implementation.

6. Logging

The AIP-validator implements the following behaviour:

- If parameters `-l` and `-o` are not set, the resulting validation report is printed to `stdout` and log messages are printed to `stderr`.
- Parameter `-o` may be used to redirect the generated validation report to the file specified as an argument
- Parameter `-l` may be used to redirect the log messages into a file specified as an argument
- Parameter -d or –debug may be used to activate verbose logging

7. Limitations

The following limitations apply:

- The AIP-eIDAS-Signature-Validator (short: AIP-Validator) has a strong dependency to the external signature validation service. The validation of electronic signatures, seals and timestamps depends on this external service. The signature verification report in the XAIP validation report is generated by the external signature verification service. The signature validation service in the current version does not support ASiC-formats. Not all variants of *AdES-compliant signatures may be verified completely by the verification service.
- Parallel XML-signatures are not yet supported for signature verification
- Validation Information for elements of type `TransformInfoType` is currently not generated due to inconsistencies in the verification report scheme
- Extensions are not evaluated due to their dependency to specific profiles
- The content of Metadata sections is not evaluated with the exception of their well-formedness

Be aware of the following issues:

- [XVAL-1] When using the parameter `-o`, the argument must refer to a file which is not in the current directory

8. Annex A: Special handling for XML signatures

The AIP-validator implements a special treatment for XML-signatures to circumvent canocalization and transformation issues. Figure 3 shows the schema of the `dataObject` element, Figure 4 shows the structure of the encoding of the signature object in an XAIP-container.

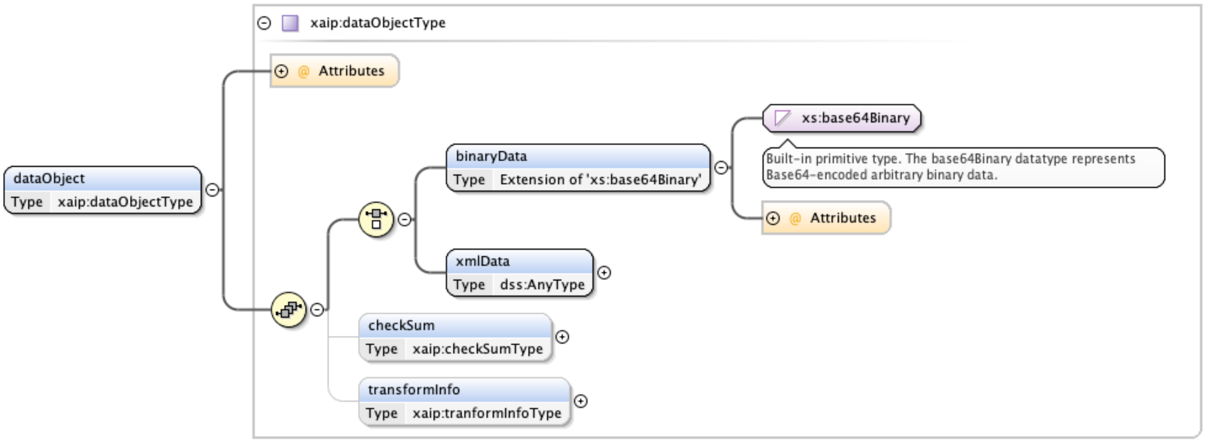


Figure 3 dataObject schema

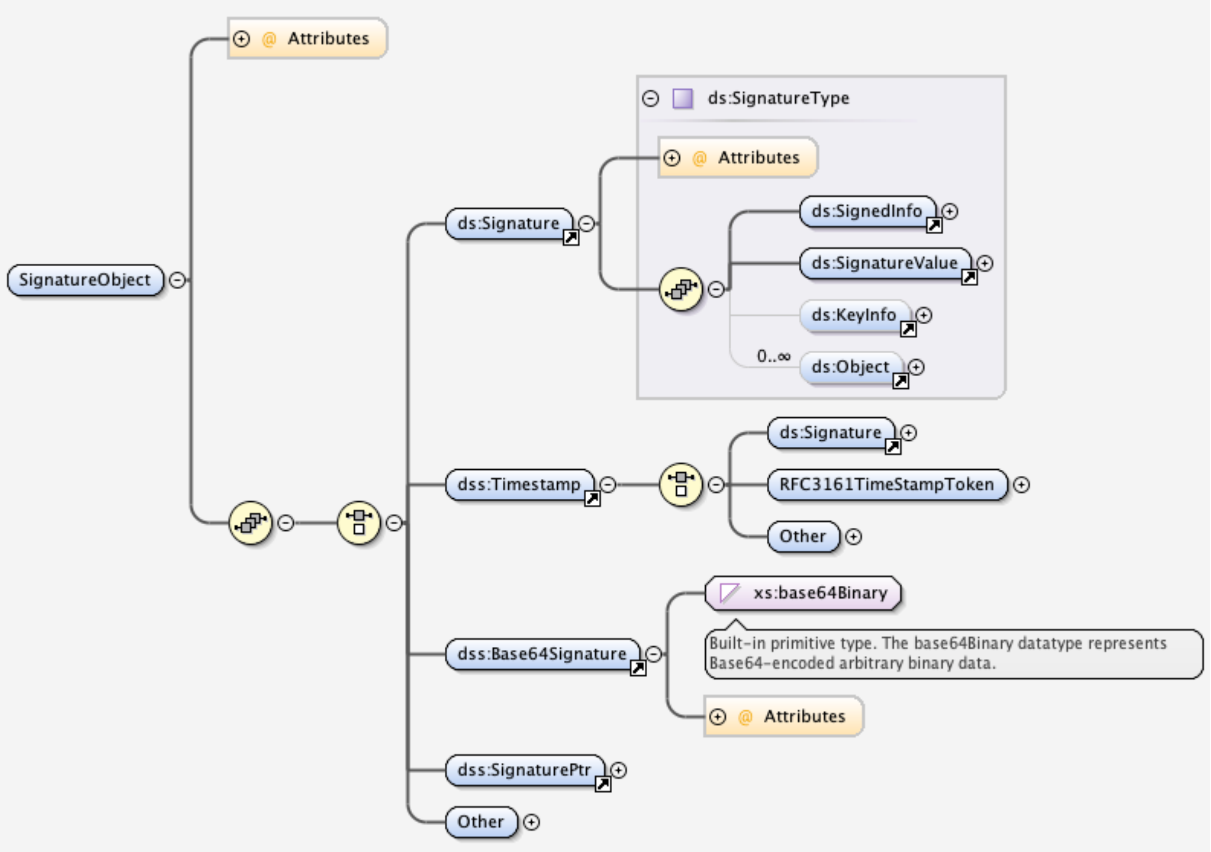


Figure 4 SignatureObject schema

Generally speaking, the AIP-validator implements the following strategy:

Enveloped and Enveloping XML-signature in xaip-element dataObjet:binaryData

The signature may be encoded using base64-encoding in the `dataObject:binaryData` element of the XAIP-container.

1) The AIP-validator decodes the data object.
2) If the result of step 1) appears to be an XML-document, the validator evaluates the XML-document for the appearance of an XML-signature object.
3) If an XML-signature object is detected, the base64-encoded document will be submitted to the signature verification service in base64-encoding using the field `dss:Base64Data` in the `VerifyRequest`

Enveloped and Enveloping XML-signature in xaip-element dataObject:xmlData

The signature may be part of a `dataObject:xmlData` element of the XAIP-container.

1) When detecting an element of type `dataObject:xmlData`, the validator evaluates the `xmlData` for the appearance of an enveloped or enveloping signature.
2) In case of a detected signature, the XML data will be submitted to the verification service in base64-encoding using the field `dss:Base64Data` in the `VerifyRequest`.

XML-signature in element xaip-element SignatureObject

1) A detached XML-signature may appear in the `dss:Base64Signature` element as part of the SignatureObject. In this case, the signature is submitted to the signature verification service in Base64 encoding.
2) A detached XML-signature may appear in element `ds:Signature` as part of the SignatureObject. In this case, the validator will extract the signature from the XAIP-object and submit the signature in Base64 encoding simulating a `dss:Base64Signature` element.
3) An enveloped or enveloping signature may appear in the `dss:Base64Signature` element as part of the SignatureObject. In this case, the signature is submitted to the signature verification service in Base64 encoding (as part of a `dss:Base64Signature` element).

9. Annex B: Supported Archival Information Package Types and Archive Data Object Types

9.1 Archivial Information Package Formats[4]

The Archival Information Package Type “XAIP” pursuant to [TR-ESOR-F], clause 3.1 is supported and in addition, the Archival Information Package Type “LXAIP” pursuant to [TR-ESOR-F], clause 3.2.

9.2 Archive Data Object Types

Furthermore, the Preservation Product may also support the Archive Data Object Types:

- CAdES pursuant to [ETSI TS 119 512] Annex A.1.1 (http://uri.etsi.org/ades/CAdES). If there is no MIME type filled, then the default application/cms is used;
- XAdES pursuant to [ETSI TS 119 512] Annex A.1.2 (http://uri.etsi.org/ades/XAdES). If there is no MIME type filled, then the default application/xml is used;
- PAdES pursuant to [ETSI TS 119 512] Annex A.1.3 (http://uri.etsi.org/ades/PAdES). If there is no MIME Type filled, then the default application/pdf is used;

[4] See also **[ETSI TS 119 512**, clause A.1.5 and A.3.2]

10. Definitions and acronyms

Abbreviation	**Keyword**
[ABC]	for: document ABC
AOID	Archive Data Object Identifier
ASiC-AIP	Associated Signature Container (ASiC)- Archival Information Package
eIDAS	REGULATION (EU) No 910/2014 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 23 July 2014 on electronic identification and trust service for electronic transactions in the internal market and repealing Directive 1999/93/EC
EU	European Union
GDPR	General Data Protection Regulation
IT	Information Technology
LXAIP	Logically XML formatted Archival Information Package
OCSP	Online Certificate Status Protocol
OID	Object Identifier
PDS	Preservation of Digital Signature
PRP	Preservation Service Protocol
PS	Preservation Service
PSP	Preservation Service Provider
PSPS	Preservation Service Practice Statement
T&C	Terms and Conditions
TL	Trusted List
TR-ESOR	DE: Technische Richtlinie zur Beweiserhaltung kryptographisch signierter Dokumente EN: Technical Guideline for Preservation of Evidence of Cryptographically Signed Documents
TS-Policy	Trust Service Policy
TSPS	Trust Service Practice Statement (see e.g. **[EN 319 401]**, chapter 6.1.)
UTC	Coordinated Universal Time
WSDL	Web Services Description Language
XAIP	XML formatted Archival Information Package
XML	Extensible Markup Language

Table 4: Keywords and Abbreviations

11. References

[eIDAS] *Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC.* OJ L 257, 28.8.2014, p. 73-114.

[ETSI TS119512] ETSI TS 119 512, *Electronic Signatures and Infrastrucutres (ESI); Protocols for trust service providers providing long-term data preservation services,* V1.1.2 (2020-10) and later versions , see https://www.etsi.org/deliver/etsi_ts/119500_119599/119512/01.01.02_60/ts_119512v010102p.pdf

[TR-ESOR-F] BSI TR 03125-F: *Preservation of Evidence of Cryptographically Signed Documents: Annex TR-ESOR-F Formats,* V1.2.1 and later versions

[TR-ESOR-VR] BSI TR 03125-VR: *Preservation of Evidence of Cryptographically Signed Documents: Annex TR-ESOR-VR: Verification Reports for Selected Data Structures*, V1.2.1 and later versions

[TR-03112] BSI TR 03112: *Technical Guideline TR-03112-1: eCard-API Framework – Overview:* V1.1.5 and later versions