Federal Office
for Information Security

BSI Technical Guideline 03125 Preservation of Evidence of Cryptographically Signed Documents

Annex TR-ESOR-EVT: Evidence Record Verify Tool (EVT)

(Conformity Level 2 – Technical Conformity)

Designation	ERVerify Tool
Abbreviation	BSI EVT
Version	1.3 (on base of the eIDAS-Regulation and the ETSI Preservation Standards with a new certification scheme)
Date	16.02.2024

Document history

Version	*Date*	*Editor*	*Description*
1.3	14.12.2022	Governikus GmbH & Co. KG[1]	ERVerify Tool for TR-ESOR Version 1.3
1.3.5	16.02.2024	Governikus GmbH & Co. KG	ERVerify Tool for TR-ESOR Version 1.3

Table 1: Document history

Federal Office for Information Security
P.O. Box 20 03 63
53133 Bonn
Tel.: +49 22899 9582- 0
E-Mail: tresor@bsi.bund.de
Internet: https://www.bsi.bund.de
© Federal Office for Information Security 2024

[1] On behalf of the Federal Office for Information Security

Table of Contents

Figures

Tables

1 Introduction

1.1 Overview

Algorithms used for signatures compliant to [ETSI_EN_319_102-1] are applicable to create eIDAS qualified electronic signatures, seals and timestamps [eIDAS] for a limited period of time only. The exact periods are continually changing and can be found, e.g. in [ETSI_TS_119_312].

Preservation systems are used in order to prolong the legal implications of the aging signatures, seals, and timestamps over an arbitrarily long period of time, i.e. the algorithms initially used are refreshed in a specific manner involving hash-trees and timestamps pursuant to [RFC4998].

This tool aims at verifying the technical evidence records (ERS – Evidence Record Syntax) according to [RFC4998] associated with an (L)XAIP, i.e. (logical) XML-formatted Archival Information Package, with a unique Archive Object ID (AOID). To achieve this, the `ERVerifyTool` verifies the partial Merkle hash-tree and sends a validation request for the timestamp via an eCard compatible interface [TR-ESOR-E] to a validation component2 to be configured separately.

1.2 Purpose

This is a tool to verify Evidence Records against protected XAIPs or LXAIPs or binary documents. It complies with [RFC4998], [EN319122-3] and TR-ESOR version 1.3. This tool allows an independent test whether Evidence Records created by a certain product comply with the requirements of [RFC4998], [EN319122-3] and TR-ESOR 1.3, especially [TR-ESOR- ERS] and [TR-ESOR-F], and thus enables interoperability between different TR-ESOR products.

2 Only if access to a proper validation service has been additionally provided and granted. The default use case does proof only the mathematically correctness of the timestamp locally.

2 Legal and other information

Although this product documentation was prepared to the best of our knowledge and with utmost care, it is not possible to completely rule out any mistakes or inaccuracies. We do not assume any legal responsibility or liability for any incorrect information, which may remain or for the consequences of such information. The information contained in this product documentation reflects the current status of development and can be changed without prior notice. Future editions may contain additional information. Technical and typographical errors will be corrected in future editions.

The ERVerifyTool itself as well as this documentation are provided under the Apache License Version 2.0 which is enclosed in the product source code distribution.

Copyright (c) 2023

Federal Office for Information Security (BSI),

Godesberger Allee 185-189,

53175 Bonn, Germany,

phone: +49 228 99 9582-0,

fax: +49 228 99 9582-5400,

e-mail: bsi@bsi.bund.de

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License.

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

3 Installation

3.1 Preconditions

The ERVerifyTool is a pure Java application and should be able to run under any Java Version 11 virtual machine. It has been tested on the following operating systems:

- Windows 10
- Ubuntu Linux 20.4

Make sure Java Version 11 in the most recent update (the application was tested on Oracle JDK 11.0.12) is installed on your system. Furthermore, set the JAVA_HOME environment variable to point to your Java installation.

Do not include any out-dated additional libraries into your Java installation. Especially, make sure no application installed a BouncyCastle of version 1.54 or older into the directory `$JAVA_HOME/jre/lib/ext`. Having an outdated BouncyCastle in your class path may cause the application to fail.

For installation and configuration, you will need a text editor and a program to unpack a ZIP archive.

To create an extension of the program, in addition to the preconditions above you will need the following:

- Java 11
- an appropriate IDE, for instance Eclipse or IntelliJ

The tool is compiled using the gradle build system, which can be used to get all other required dependencies from public online repositories. The gradle build system does not need to be installed, it can be used through the enclosed wrapper (see README.md for more information).

3.2 Build instructions

The `ERVerifyTool` is provided as open source software and has to be built from scratch. The source of the `ERVerifyTool` can be obtained from the corresponding git-repository https://github.com/de-bund-bsi-tr-esor/ERVerifyTool either by downloading a ZIP-package or by using the cloning functionality of the git tool.

In order to build the `ERVerifyTool` following several steps have to be performed:

1. Change to `ERVerifyTool` directory of the sources
2. Execute following command, depending of the used operating system:
   a. Windows: `.\gradlew clean build -Prelease -DskipIntegrationTests`
   b. Linux: `./gradlew clean build -Prelease -DskipIntegrationTests`

The built `ERVerifyTool` can be found under following paths:

1. as a zip archived distribution file under: `ERVerifyTool/all/build/dists/ERVerifyTool-all-*-bin.zip`, or
2. as a directory structure under: `ERVerifyTool/all/build/install/*`.

In order to use the tool, the configuration steps have to be performed in advance. Please refer to section 4.1 for further information.

3.3 Command Line Application

After unpacking the distribution ZIP file (c.f. section 3.2), the command line application does not need any further installation. It is started by calling the script `checktool` (for Linux) or `checktool.cmd` (for Windows), respectively, in the directory `ERVerifyTool/cli/bin`.

In case of Linux, you may need to make the script executable by calling `chmod u+x checktool`.

Before using the application, you have to create a valid configuration. See chapter 4.1 for further details

3.4 Standalone Web Service

The command line application has to be set up in advance (c.f. section 3.3). To start the web service, configure the application (c.f. section 4.1) and call `checktool -server -port <PORT> -conf <FILE>` where `<PORT>` is the number of the port to listen on and `<FILE>` is the name of the configuration file.

With Windows, call `checktool.cmd` with same parameters. Be aware that you need root or administrator privileges, respectively, in case the port is less than 1024.

3.4.1 Example of the starting-call in case of windows

Call: `checktool.cmd -server -port 8080 -conf ..\config\config-rfc4998-of-fline.xml`

3.5 Web Service in Tomcat

Install Apache Tomcat version *10.0* (tested with 10.0.16) by following the official Tomcat installation instructions on a Java *11* JDK distribution.

To deploy the `ERVerifyTool` you copy the WAR file `war/ErVerifyTool-<version>.war` into the `$CATALINA_HOME/webapps` directory after renaming it to `ErVerifyTool.war`. If you do not rename the WAR file beforehand the address of the deployed application may differ (see below).

Create a configuration of the `ERVerifyTool` application as described in the chapter 4.1. Copy that file into the directory `$CATALINA_HOME/bin/conf` and name it `ErVerifyTool.xml`. Furthermore, create a valid Log4J2 configuration `log4j2.xml` in the same directory.

In case you want to run multiple instances of the `ERVerifyTool`, with different configurations within the same server, you may pack a valid configuration inside the `.war` file. Place a configuration file named `ErVerifyTool.xml` inside the `.war` file under `WEB-INF/classes`. If no configuration is packed within the `.war` file, the application searches in the `$CATALINA_HOME/bin/conf` folder of the Tomcat for a configuration named `ErVerifyTool.xml`.

The web application is reachable on a local machine with default Tomcat port at[3]:

`http://localhost:8080/ErVerifyTool/`

3 In case the default tomcat configuration is used.

That overview page provides a link to the TR-ESOR S.4 web service as well as information about the loaded configuration.

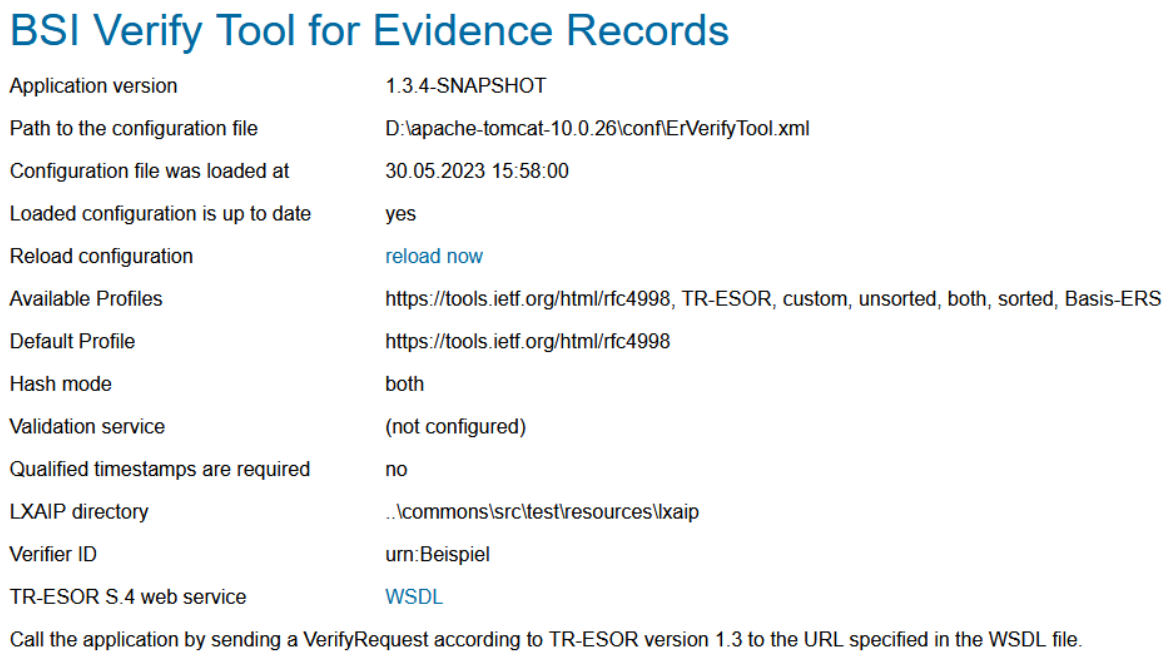

Figure 1: Example view of the configuration overview page for Tomcat deployment

The configuration can be reloaded by using the link “reload now” (e.g. after a new version of the configuration has been placed on the server). A Service WSDL can be obtained by using the link “WSDL” depicted on the figure. Furthermore, presents the generated site some general information about currently used configuration, e.g. last reload date or used profile.

4 Usage

4.1 Configuration

The configuration is contained in an XML file, which can be edited with any text editor. The schema for creating configuration files can be found in the config directory.

Edit the enclosed file `config/config.xml` to match your requirements. The following properties can be specified in the General-section:

- VerifierID (mandatory): The ID of the verifier to appear in the verification report. Choose any URI which describes your installation and configuration of the `ERVerifyTool`.
- DefaultProfileName (mandatory): URI to define the profile that will be used by the command line calls or by web service calls which do not explicitly specify another profile. Allowed values are any profiles you specify in the following section and the built-in profiles
  - `https://tools.ietf.org/html/rfc4998`
    This profile checks according to RFC4998. The qualification status of timestamps is not checked and additional requirements from TR-ESOR ERS are not checked.
  - `TR-ESOR`
    In addition to the checks performed for RFC4998, this profile checks the qualification status of timestamps according to eIDAS and requires all timestamps to have the qualified status. This requires an eCard service that provides SignatureQuality statements.
  - `Basis-ERS`
    This profile performs all checks required for RFC4998 and additional checks according to the TR-ESOR ERS annex. This especially includes checks of the content info included in the timestamps.

The following settings are needed only in case you want to add a plug-in (Validator, Parser or HashCreator) or want to use some application part other than the default.

- `General/ConfiguredObjects` (optional): Here you may specify plug-ins which are applicable in all supported profiles. These settings may be replaced by special definitions in a `Profile`-section.
- `Profile` (optional): This part may occur several times to define new profiles and objects which replace the defaults for the respective profile. A profile consists of
  - `name` (mandatory): A name (URI)
  - `hashMode` (optional): Property to set the hash concatenation mode for rehashed evidences:
    - `unsorted` (default): do not sort hashes, according to RFC 4998, section 5.2, point 4
    - `sorted`: use binary sorted hashes as specified in RFC6283, section 4.2.2, point 6. This mode might be selected for ASN.1-based evidence records as well, but evidence records are not completely compliant to RFC4998 if this mode is used
    - `both`: accept both modes
  - `validationService` (optional): This is a URL to the WSDL of an eCard compatible validation web service. This is needed for a comprehensive check of timestamps and digital signatures. If omitted, the certificate chain and further details of timestamps and signatures cannot be checked and the overall check result cannot be better than indetermined[4].

4 Only mathematical correctness will be checked.

- `verifySignatures` (mandatory): This attribute can be set to `true` or `false`. When set to `true` signatures / seals are verified. When set to `false` signature / seal verification is omitted in the validation process.
- `requireQualifiedTimestamps` (optional): Might be set to `true` or `false`. This value can be set to `true` to require timestamps to be checked as qualified according to a European Union Memberstate Trusted List. This requires the online validation of timestamps to be active by using a profile with a configured validation service URL and the ECardTimeStampValidator configured. Enabling this option requires the validation service to provide a SignatureQualityType according to the ETSI Signature Validation Report to be present in the details returned for a timestamp check. If omitted, this defaults to `false` and the qualification status does not impact the overall result.
- `lxaipDataDirectory` (optional): The path to the directory LXAIP's data object references should resolve to. The data object reference's URI is then resolved relative to the given directory. If omitted, LXAIPs will not be validated.

Within the `Profile` and `General/ConfiguredObjects` sections you may specify a validator which handles a certain type of parsed object. Any configured validator replaces the respective default validator which is built into the application itself. Settings for the profile overwrite general settings.

A `Validator` element is defined with the following values:

- `className` (mandatory): The fully qualified name of the validator class
- param (optional, may occur several times): Name and value of a constructor parameter. Parameter type must be String.
- `targetType` (mandatory): Fully qualified class name of objects that the validator can handle. If two validators are defined in the same section, one targeting a specific target type and the other some base class, the application will choose the one for the most specific type which matches the object to be validated. Both validator class and target class must be present in the class path. Target types occurring in the current version of `ERVerifyTool` without additions (see "Appendix: Internal data types" for more details) are:
  - `org.bouncycastle.tsp.TimeStampToken`
    This is the RFC3161 compliant timestamp. A validator can be implemented to provide a comprehensive report. The ECardTimestampValidator is an example of a configurable validator for TimeStampToken.
  - `de.bund.bsi.tr_esor.checktool.data.AlgorithmUsage`
    This is the algorithm used in the hash tree. A validator for AlgorithmUsage must be able to check wether a specific algorithm was accepted as a secure algorithm on a specific day. The default implementation is based on an algorithm catalog enclosed in the tool.
  - `de.bund.bsi.tr_esor.checktool.data.ArchiveTimeStamp`
    This is the ArchiveTimeStamp. The default implementation checks the mathematical correctness of the partial hashtree and uses the validator configured for TimeStampToken to check the enclosed timestamp.
  - `de.bund.bsi.tr_esor.checktool.data.ArchiveTimeStampChain`
    This is the archive timestamp chain. The default validator uses the validator configured for ArchiveTimeStamp to check the enclosed ArchiveTimeStamps and checks for the cryptographic integrity of the chain.
  - `de.bund.bsi.tr_esor.checktool.data.ArchiveTimeStampSequence`
    This is the archive timestamp sequence. The default validator uses the validator configured for ArchiveTimeStampSequence to check the enclosed ArchiveTimeStampChain and checks for the cryptographic integrity of the sequence.

- `de.bund.bsi.tr_esor.checktool.data.EvidenceRecord`
  This is the complete evidence record. You can reimplement or enhance the validator to provide additional checks regarding the complete evidence record.

In the General section you may additionally specify a hash creator. Default is local hashing, you might want to use some certified crypto module instead. Furthermore, that section allows you to define name space prefixes for XML serialization by adding `NamespacePrefix`[5] elements. This may be necessary because web service access might disregard the prefixes used in a given (L)XAIP. Define an empty prefix to use as the target (prefix-less) name space. **Default name space prefixes are as defined in TR-ESOR XAIP V1.3 schema**.

Adding namespace prefixes is only required if the namespaces used in a(n) (L)XAIP differ from the default namespaces, if default namespaces are used, no adjustments are required.

For example, if a namespace such as `xmlns:esor=`http://www.bsi.bund.de/tr-esor/xaip is declared in a(n) (L)XAIP, the namespace binding looks like this:

```
<NamespacePrefix namespace="http://www.bsi.bund.de/tr-esor/xaip">
   esor
</NamespacePrefix>
```

The current configuration schema allows defining additional parsers. However, because the application already contains all necessary parsers for the currently supported use cases, you do not have to specify any further parsers.

Adding a validator: Add the library containing your validator to the application class path. Add a `Validator` element to the appropriate part of the configuration.

Removing a validator: Remove the respective `Validator` element from the configuration.

Listing the configured validators: Read the configuration file.

Checking the correctness of the configuration file: This is done automatically when you start the application. In case of problems, the application will terminate immediately and write an appropriate message to standard output.

Examples of configuration files

For example, the following configurations can be created:

- config-BasisERSProfile-offline

  This configuration as an example looks like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Configuration xmlns="http://www.bsi.bund.de/tr-esor/checktool/1.2">
  <General>
    <VerifierID>ERVerifyTool for Basis-ERS</VerifierID>
    <DefaultProfileName>Basis-ERS</DefaultProfileName>
  </General>
  <Profile      name="Basis-ERS"      lxaipDataDirectory="/data/lxaip/"
validationService="https://validation-
service.de/validation/eCard?wsdl" requireQualifiedTimestamps="true">
    <Validator>
<className>de.bund.bsi.tr_esor.checktool.validation.default_impl.basis
.ers.BasisErsECardTimeStampValidator</className>
      <targetType>org.bouncycastle.tsp.TimeStampToken</targetType>
    </Validator>
```

[5] The `NamespacePrefix`-element can be used only in combination with SOAP-interface (c.f. chapter 4.4).

```
  </Profile>
</Configuration>
```

- config-rfc4998-offline – Validation of the syntax of an Evidence Record according to the RFC4998-Profile, no online-validation of timestamps (the whole process offline). This can use the default profile, so that only the general section of the configuration needs to configured.
- config-rfc4998-online. - Validation of the syntax of an Evidence Record according to the RFC4998-Profile and online-validation(s) of the timestamps in the Evidence Records via a special validator on base of a validation service using the eCardAPI-Interface. This needs the `ECardTimeStampValidator` and a validation service URL to be configured.

  This configuration as an example looks like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Configuration xmlns="http://www.bsi.bund.de/tr-esor/checktool/1.2">
  <General>
    <VerifierID>ERVerifyTool for RFC4998 with online check</VerifierID>
<DefaultProfileName>https://tools.ietf.org/html/rfc4998</DefaultProfil
eName>
  </General>
  <Profile                    name="https://tools.ietf.org/html/rfc4998"
lxaipDataDirectory="/data/lxaip/"
validationService="https://validation-
service.de/validation/eCard?wsdl" requireQualifiedTimestamps="false">
    <Validator>
<className>de.bund.bsi.tr_esor.checktool.validation.default_impl.ECard
TimeStampValidator</className>
      <targetType>org.bouncycastle.tsp.TimeStampToken</targetType>
    </Validator>
  </Profile>
</Configuration>
```

- `config-qualified-timestamps-online`: Profile checking Evidence Records requiring qualified timestamps. Requires an online eCardAPI validation interface providing information on the timestamp quality. This is equivalent to the default TR-ESOR profile.

Multiple profiles might be specified in a single configuration file and selected through the profile switch of the command line application. The web service uses the default profile specified in the general section.

The profile Basis-ERS contains a number of custom validators that perform additional checks in comparison to the RFC4998 base profile. If a validation based on **[TR-ESOR-ERS]** should be performed, the profile *must* be named Basis-ERS in the configuration as well. All other profiles are based on the RFC4998 profile. For online validation, the BasisErsECardTimestampValidator must be used to comprehensively check timestamps for the Basis-ERS profile.

4.2 General Validation

Validating evidence records can be done via command line application or via web service. In both cases, the evidence record may be given separately, embedded within a(n) (L)XAIP or embedded as unsigned attribute within a CMS signature. Evidence records within a(n) (L)XAIP or CMS structure are only recognized if they are embedded correctly as specified in TR-ESOR 1.3 or CAdES pursuant to [EN319122-3], respectively.

If an evidence record is given but no protected data is passed to the application, only the internal structure of the evidence record will be validated. If no evidence record is embedded within the given (L)XAIP and no evidence record is given separately, an empty verification report is returned.

Profiles supported by the application are:

- https://tools.ietf.org/html/rfc4998
- Basis-ERS
- TR-ESOR (requires online validation of qualified timestamps)
- Any further profiles specified by administrator in the configuration.

More Information on the default profiles can be found in section 4.1.

4.3 Calling the Command Line Application

With parameter -h or in case of invalid parameters, the application just displays a help message.

The tool can be started using the `checktool[.bat]` command in the bin folder of the CLI distribution. After building the tool, the CLI distribution might be found as a zip and tar compressed folder in the path ERVerifyTool/cli/build/distributions.

Arguments	Description
`-conf <arg>`	path to the configuration file
`-data <arg>`	path to the file containing the secured data (optional) if parameter -er is specified), if omitted, the ER will be validated in itself but result will be indetermined at best.
`-er <arg>`	path to the file containing the evidence record(optional)
`-host <arg>`	hostname for the server mode, default is localhost
`-out <arg>`	path to the output folder (optional, default is standard out). A new folder will be created with the name of the aoid. The combined report and the signed data of a given (L)XAIP will be saved.
`-port <arg>`	listen port for server mode, defaults to 9999
`-profile <arg>`	name of the profile to use for verification (optional, default is https://tools.ietf.org/html/rfc4998), see Section Configuration (4.1) for more information about profiles
`-server`	start as web service (optional, ignores all other parameters except -conf and -port)

Table 2: Command line options for calling the tool on the CLI

If the data parameter is a file containing a XAIP, then the evidence records and contents that might contain signatures embedded in that XAIP and optional electronic signatures and seals themselves are checked as well. LXAIPs containing `DataObjectReferences` can also be provided as data. The references can only be resolved from files in the configured LXAIP data directory (see Section 4.1).

When a(n) (L)XAIP is provided as input, there must be an exact one to one mapping between the protected objects in the XAIP and the hash values present in the first partial hash tree of the evidence record.

XAIPs of the previous versions like TR-ESOR 1.1 or 1.2 or 1.2.1 or 1.2.2 are not supported. If those older XAIPs are used as data, the application cannot check the correctness of the hash values included in the XAIP. Older versions of this tool might be used to check XAIPs according to TR-ESOR 1.2, 1.2.1 and 1.2.2. The version might be obtained from the V1.2.2-Branch[6] of the GitHub repository. All other formats given as `data` parameter are handled as binary protected content and are not interpreted in any way. Binary contents must be

[6] See https://github.com/de-bund-bsi-tr-esor/ERVerifyTool/tree/V1.2.2

exactly as hashed during the evidence record creation and any changes will lead to a mismatching hash value. The file given as parameter `er` may contain:

- an ASN.1 evidence record
- an XML with root tag `{http://www.bsi.bund.de/tr-esor/xaip}:evidenceRecord` containing an ASN.1 evidence record. In this case the application will fail if the data parameter does not contain an XAIP with specified AOID and version.
- a CMS signature with embedded evidence records (CAdES-E-ERS)

If a XAIP has been migrated from an older TR-ESOR version through the container in container migration approach described in TR-ESOR 1.3 M2 Section 2.6, the whole new container needs to be provided for a hash value check. A check of the old XAIP with the evidence record constructed for the new 1.3 XAIP container is not supported.

To verify evidence records and optionally signatures in XAIPs, typically call:

```
checktool[.bat] -conf <FILE> -data <XAIP or bin file> [-er <detached evidence record>] [-out <FILE>]
```

The output of the validation will be a verification report with all checked details.

If the -out parameter is used to provide an output folder, all results are written to the folder specified. A subfolder is created with the AOID of the (L)XAIP as name. If no AOID is available, the output is written to the folder no_aoid. Inside that folder, a subfolder for each data object and each credential is created. As (L)XAIP-containers do not preserve the original filename or filename extension, you might need to manually add a filename extension or select an appropriate application to open the exported file.

Optionally: For signatures, the signed data is written to the output folder as well.

To start the stand-alone web service, call:

```
checktool[.bat] -conf <FILE> -server -port <PORT>
```

4.3.1 Logging

By default, the command line application will output logging information onto the command line. To change the logging configuration of the tool, set the environment variable `LOG4J_CONFIGURATION_FILE` to the path of a log4j2 configuration file. If the example logging configuration file located in `cli/src/main/resources/log4j2.xml` is used, a log file named ERVerifyTool.log is created in the working directory. See https://logging.apache.org/log4j/2.x/manual/configuration.html#XML for further details.

4.4 Calling the Web Service

The service WSDL is identical to the one defined in TR-ESOR version 1.3. Only the VerifyRequest is supported here. The web service requires no authentication and can be invoked by any appropriate web service client.

The service WSDL can be reached at the following URL:

```
http://<HOST>:<PORT>/ErVerifyTool/esor13/exec?wsdl
```

If the application is deployed on Apache Tomcat as a war file, additional information is displayed at

```
http://<HOST>:<PORT>/ErVerifyTool
```

Inside the verify request, the data to check must be provided under the following XPaths.

Element	XPath
detached evidence record	/VerifyRequest/SignatureObject/Base64Signature or /VerifyRequest/SignatureObject/Other/evidenceRecord/asn1EvidenceRecord
binary protected data elements	/VerifyRequest/InputDocuments/Document/Base64Data
(L)XAIP which may contain embedded evidence records and signatures as XML	/VerifyRequest/InputDocuments/Document/InlineXML/ XAIP
(L)XAIP which may contain embedded evidence records and signatures as encoded XML	/VerifyRequest/InputDocuments/Document/Base64XML
CMS signature with embedded evidence records	/VerifyRequest/SignatureObject/Base64Signature

Table 3: Positioning of data contents in web service requests

When validating a detached evidence record or a detached CMS signature with embedded evidence records, you should specify all protected data elements or the addressed XAIP, respectively, as input document(s). Otherwise, the tool checks only the internal structure of the evidence record itself.

LXAIPs are provided in the same fields as XAIPs. When a LXAIP is provided, the `DataObjectReferences` in the LXAIP need to be resolvable in the local file system directory configured in the `lxaipDataDirectory`-Attribute of the applied profile. Otherwise, LXAIPs are handled the same way as XAIPs and may be included in the same fields

Signatures from a(n) (L)XAIP can only be checked if an online validation service is configured for the active profile and `verifySignatures` is set to "`true`"[7]. In this case, the documents will be extracted from the respective fields in the (L)XAIP and all signatures, seals, and timestamps found (apart from archive timestamps) will be validated depending on the validation service used. Signatures that are not included in a(n) (L)XAIP should be checked directly by a validation service. The following signature types have been tested: all baseline profiles of CAdES, JAdES, PAdES, XAdES and ASiC as well as RFC3161-compliant timestamps.

The evidence data for a(n) (L)XAIP can only be checked if the XML structure including any namespace prefixes can be properly reconstructed. When embedding a(n) (L)XAIP into the request as Inline-XML, namespaces are not preserved through the web service call and will be replaced with the namespaces configured in the General section of the configuration or the namespace prefixes as given in the BSI TR-ESOR 1.3 schema. Using a non-exclusive canonicalization algorithm might lead to problems with this kind of embedding a(n) (L)XAIP into the request as additional namespaces might be incorrectly added into the canonicalized elements during hash value checks.

Providing a(n) (L)XAIP as `Base64XML` preserves the original namespaces and some other structural information of the XML like line breaks and is recommended.

If an evidence record or a CMS signature is given as value of `/VerifyRequest/ SignatureObject/Base64Signature`, then the application will detect the type of the given object by analysing the value itself.

Furthermore, the request usually should contain an optional input of type:

[7] See clause 4.1

`{urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#}ReturnVerificationReport`

to cause the application to return a verification report. Without that optional input the response will only contain a result with technical information whether the request was processed, but not the result of the validation. In most cases you should set the value of attribute `ReportDetailLevel` to

`urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:reportdetail:allDetails.`

Other allowed values are:

- `urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:reportdetail:noDetails`
- `urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:reportdetail:noPathDetails` (line breaks are not part of the values)

If the validation of the evidence record should be done using another profile than the default profile specified in the configuration, the attribute `profile` of the `VerifyRequest` must be set.

4.5 Verification Report

The tool will output an XML-based verification report according to OASIS DSS for all evidence records provided and in case of `verifySignatures`="`true`" also for all electronic signatures of seals.

Every checked object will have an own, individual report embedded into the overall verification report. Every report contains a `SignedObjectIdentifier` giving information on the part of the input data covered by the report. For example, if a(n) (L)XAIP with detached ER is checked on the cli, the `SignedObjectIdentifier` might be "command line parameter `data/evidenceRecord:ER-V001`".

4.5.1 Online Validation

When the online validation of timestamps or (optionally) electronic signatures or seals through an eCard service is configured, the other content of the individual reports is derived directly from the eCard service including all result codes and messages. The results from an online validation might therefore differ from those provided through an offline check. The tool has been tested to work with vendor-specific validation software as eCard service. The validation model used, i.e. shell, chain, or hybrid, depends on the validation policy of the configured (external) Validation Services, which are installed to validate the electronic timestamps or (optionally) signatures or seals.

An example report can be found in `doc/examples/report in the source code repository`[8].

4.5.2 Offline Validation

In case of `verifySignatures`="`true`", when used offline, the tool will not perform any checks whatsoever on signatures or seals. The result of a signature validation will always have the result major "`urn:oasis:names:tc:dss:1.0:resultmajor:InsufficientInformation`" and the result minor "`http://www.bsi.bund.de/tr-esor/api/1.3/resultminor/arl/notSupported`"

[8] See https://github.com/de-bund-bsi-tr-esor/ERVerifyTool/tree/master/doc/examples/report.

For timestamps inside of evidence records, the format and the mathematical correctness is checked, but the certificate path is not checked. The result for the certificate path check is always undetermined when an offline check is performed.

An example report can be found in `doc/examples/report` in the source code repository.[9]

[9] https://github.com/de-bund-bsi-tr-esor/ERVerifyTool/tree/master/doc/examples/report

5 Digital Signature Verification

Notice 1

Digital signatures and timestamps from a(n) (L)XAIP (apart from archive timestamps) can only be verified if an online validation service is configured and running for the active profile and the optional parameter "`verifySignatures`" is set to "`true`".

Notice 2

Digital Signature Verification with this `ERVerifyTool` is currently not subject to the conformity assessment of products against TR-ESOR in version 1.3.

Furthermore, the

- **tr-esor-AIP-eIDAS-SigValidator**[10]

is used for those purposes by the TR-ESOR-C.2-Testbed[11] currently. Therefore, in case of TR-ESOR conformity assessments the attribute "`verifySignatures`" shall be set to "`false`".

Additionally, to checking Evidence Records, the tool can also extract signatures, electronic seals and document-related timestamps from (L)XAIPs and send them to an eCard-compatible validation service for verification in case that `verifySignatures` is set to "`true`"[12]. The web service needs to be configured using the validation service attribute of the profile (see Section 4.1). Without the online service, all data objects and signatures will be marked as `indetermined` in the reports. If no online validation service is configured, the `ResultMajor` is `urn:oasis:names:tc:dss:1.0:resultmajor:InsufficientInformation`, the `ResultMinor` is `http://www.bsi.bund.de/tr-esor/api/1.3/resultminor/arl/notSupported` and the message explains the situation: "No online validation of a potential signature was possible as no validation service is configured in the active profile."

For each data object and each signature credential inside a(n) (L)XAIP, an individual report is included in the Verification Report pursuant to **[OASIS DSS]**[8] provided by this tool. The contents of this report are directly derived from the validation service. An eCard-compatible validation service should provide individual reports according to OASIS DSS, which are included in the DSS compliant overall verification report.[13]

When an online verification service is configured, the tool will submit every data object, meta data object, signature and document-related timestamp to the validation service. The supported signature formats are therefore only limited by the verification component and might include CAdES, PAdES, XAdES, JAdES, ASiC, RFC3161 compliant timestamps and more.

The verification report aggregates all reports provided and provides information on all objects that do not contain a signature as well. The report for the single unsigned data object contains a requester error as result major, but the overall result of the report is not affected by unsigned data objects or meta data elements. Unsigned data objects will get the ResultMajor code `urn:oasis:names:tc:dss:1.0:resultmajor:Success` and the Result Message "No inline signature found in data object. Detached signatures might be present." as only contents of their report. For an example report see doc/examples/report in the source code repository.[14]

[10] See https://github.com/de-bund-bsi-tr-esor/tr-esor-AIP-eIDAS-SigValidator
[11] See https://github.com/de-bund-bsi-tr-esor/TR-ESOR-C.2-Testbed
[12] See clause 4.1
[13] See example in https://github.com/de-bund-bsi-tr-esor/ERVerifyTool/tree/master/doc/examples/report
[14] See https://github.com/de-bund-bsi-tr-esor/ERVerifyTool/tree/master/doc/examples/report

In order to be correctly detected as report for unsigned data, the eCard service is expected to either return a result major “ok” with an empty report or a result major “error” with the minor code http://www.bsi.bund.de/ecard/api/1.1/resultminor/il/signature#signatureFormatNotSupported.

The EvidenceRecordReport and the individual reports for the signatures are not collected into a XAIPReport as this would include additional verification steps for the structure of the (L)XAIP container that are currently not supported by this tool.

Additionally, if an output folder is specified (see Table 2), the signed data objects and the detached signatures are extracted from the (L)XAIP and written into the output folder.

An example call for validation of a signature inside a XAIP on the command line interface looks like this:

```
./checktool -conf config.xml -profile TR-ESOR verifySignatures=”true” -data
../xaip/xaip_ok_sig.xml -out verifyResults
```

As a result of this call, the output folder verifyResults will be created in the current working directory. In this folder, an output folder with the AOID of the XAIP provided or no_aoid in case no AOID can be found will be created. Inside this folder, there will be a report.xml aggregating the reports for all signatures and evidence records present. In addition, a subfolder for each data object and signature is created and the contents of the data objects and signatures is written into the output folder.

The report contains an individual report for each data object and signature, even if it does not contain any signatures.

6 Creating an Additional Validator

You may extend the verification logic of the application by writing own classes for verifying certain objects.

As an example, the application already contains two validator classes for validating RFC 3161 timestamps, namely one which calls an external application to do online verifications of time stamp certificates and a `DummyTimeStampValidator`, which does not require any online connection. The

DummyTimeStampValidator is the default. It is used to validate timestamps unless some other validator is specified.

To activate the eCard base timestamp validator, include the following tag into the `General/ConfiguredObjects` section of your configuration:

```
<Validator>
      <className>de.bund.bsi.tr_esor.checktool.validation.default_impl.ECardTimeStampValidator</className>
      <targetType>org.bouncycastle.tsp.TimeStampToken</targetType>
</Validator>
```

See appendix A for the list of built-in validators and respective target classes.

6.1 How the Application chooses Validators

Whenever a certain parsed object is to be validated, an appropriate validator object is requested from the `ValidatorFactory`. That factory knows the verification profile of the current context. First, it looks for validators which are mentioned in the respective `profile` section of the configuration. If that section contains more than one validator with a matching target class (which may be some base class or interface of the object we are about to validate), it chooses the most direct one. If the `profile` section of the configuration does not contain a matching validator, then the `General/ConfiguredObjects` section is searched in the same way. If still no suitable validator is found, the factory uses the same rules for selecting one of the built-in validators.

Furthermore, every call to the `ValidatorFactory.getValidator` method must provide a `ValidationContext` object and may request a validator which creates a certain type of report. The factory restricts its search to all validators which can work with the given context and can create the requested report type.

6.2 Writing a Validator

Use the SDK to provide the necessary classes in your class path. The provided libraries contain the whole ERVerifyTool. Access is provided to all existing classes to call or inherit. The API documentation is available as appendix in this document or in HTML format in the directory `sdk/apidocs`.

6.2.1 Interface and Base Class

Write a class implementing the interface `de.bund.bsi.tr_esor.checktool.validation.Validator`. Closely follow the requirements within the API documentation of each respective interface or base class. In general, you should consider extending the class

`de.bund.bsi.tr_esor.checktool.validation.default_impl.BaseValidator`

which provides some basic checks to ensure that validation parameters and context match. The validator must have a constructor without parameters or one with a single parameter of type `java.util.Map<java.lang.String,java.lang.String>`.

The `validate` or `validateInternal` method expects the parameters:

- `ref` - a unique reference to identify the checked object
- `toBeChecked` - the object itself

That method should validate the object and return an instance of

`de.bund.bsi.tr_esor.checktool.validation.report.ReportPart` ,

which contains the validation results.

The `Validator` interface is a generic class with type parameters

- type of object to validate
- type of `ValidationContext` that class can work with. If you do not have any requirements to that context, specify the type `ValidationContext` itself.
- type of `ReportPart` created by the validator.

It is strongly recommended to validate only one level of object in a validator and delegate validation of sub-objects to other special validators. To obtain further validator instances, always call the

`de.bund.bsi.tr_esor.checktool.validation.ValidatorFactory.getValidator`

method specifying class of object to validate, class of report part to create and current context.

As an example, see class

`de.bund.bsi.tr_esor.checktool.validation.default_impl. ECardTimeStampValidator.`

During validation, the validator may assume that the method `setValidationContext` has been called previously. Thus, information from validating other parts of an object tree usually is available. Currently, the application only uses `ValidationContext` objects of class

`de.bund.bsi.tr_esor.checktool.validation.ErValidationContext`

which contains for instance information about which hash values must be covered.

6.2.2 The Validation Context

During validating a more complex object like an evidence record, certain `Validator` objects may need access to data or validation results regarding other parts of the object structure. This data is collected in an object called validation context, which is available throughout validation of the whole structure to all validators. Each validator must specify which kind of `ValidationContext` it can work with.

When calling another validator from within a validator, normally the current context is passed.

6.2.3 The Reference

All validated objects within an object tree are addressed by an instance of

`de.bund.bsi.tr_esor.checktool.validation.report.Reference.`

The references contain at least a human-readable field name, which is useful for debugging purposes. Furthermore, the reference may contain other information to be used in XML verification reports. When calling the validation of some sub-object, you should create an own reference for that object by calling `Reference.newChild(String).`

6.2.4 Parameter and Return Types of Validation

Objects which are passed as parameter `toCheck` to a `Validator.validate` method will have one of the types listed in appendix “Internal data types” or may have an additional type if

- the application is extended to validate other given objects, for instance to create (L)XAIP reports
- an added validator encounters another object within the object it is validating and decides to delegate the validation of that sub-object.

All existing subclasses of `de.bund.bsi.tr_esor.checktool.validation.report.ReportPart` are supported by the generator for the XML verification report. If you decide to write your own `ReportPart` class, you should let it implement

`de.bund.bsi.tr_esor.checktool.validation.report.OutputCreator<T>`.

Because an XML verification report is currently the only supported output type, it is always possible to satisfy the needs of output creation by implementing `OutputCreator<IndivudualReportType>`.

6.2.5 Adding Your New Validator

Depending on whether your validator is specific to a certain profile or usable with all supported profiles, declare the new validator in the respective section Profile or in section `General/ConfiguredObjects`. Within the `Validator` tag, you have to specify

- the fully qualified name of the validator class
- the fully qualified name of the data class it can validate
- in case it requires construction parameters (Map), all entries for that parameter map

Add the new validator class to the class path and start the command line application providing the parameter `-conf <filename>` only. The application will check whether the configuration has correct format and all validators can be created properly.

7 Annex A: Internal Data Types

The ERVerifyTool provides default validators for the following types of objects to validate:

- `de.bund.bsi.tr_esor.checktool.data.AlgorithmUsage`
- `de.bund.bsi.tr_esor.checktool.data.ArchiveTimeStamp`
- `de.bund.bsi.tr_esor.checktool.data.ArchiveTimeStampChain`
- `de.bund.bsi.tr_esor.checktool.data.ArchiveTimeStampSequence`
- `de.bund.bsi.tr_esor.checktool.data.EvidenceRecord`
- `org.bouncycastle.tsp.TimeStampToken`

See the comments included in the source code of the respective classes for further information.

This list may be extended in the following cases:

- A new parser is added which produces another type of object to validate.
- A new validator is added which encounters another type of sub-object that wants to delegate its validation to another validator taken from the factory.

The list of built-in validators is as follows.

- Validators for each profile (unless specified otherwise in the respective profiles) are all in package `de.bund.bsi.tr_esor.checktool.validation. default_impl`
- AlgorithmUsageValidator
- ArchiveTimeStampChainValidator
- BaseValidator
- EvidenceRecordValidator
- ArchiveTimeStampSequenceValidator
- ArchiveTimeStampValidator
- DummyTimeStampValidator
- Validators for ERS basis profile (in sub-package `basis.ers`)
    - BasisErsAlgorithmUsageValidator
    - BasisErsDummyTimeStampValidator
    - BasisErsArchiveTimeStampChainValidator
    - BasisErsArchiveTimeStampSequenceValidator
    - BasisErsEvidenceRecordValidator
    - BasisErsArchiveTimeStampValidator

Furthermore, the application contains the classes

`de.bund.bsi.tr_esor.checktool.validation.default_impl.EcardTimeStampValidator`

and

`de.bund.bsi.tr_esor.checktool.validation.default_impl.basis.ers.BasisErsECardTimeStampValidator`

for online validation of time stamps (`org.bouncycastle.tsp.TimeStampToken`) by calling an external eCard-API service, for instance Governikus Suite. Those two classes have to be declared in the configuration to be used.

More precisely, insert the following block into a Profile with configured `validationService`.

<Validator>

```
      <className>de.bund.bsi.tr_esor.checktool.validation.de-
fault_impl.ECardTimeStampValidator</className>
      <targetType>org.bouncycastle.tsp.TimeStampToken</targetType>
</Validator>
```

Both validators `ECardTimeStampValidator` and `BasisErsECardTimeStampValidator` require an eCard-compatible validation service and have been tested with Governikus DATA Varuna (Part of the Application of the German IT Planning Council) as service provider.

8 Definitions and acronyms

Abbreviation	Keyword
[ABC]	for: document ABC
AOID	Archive Data Object Identifier
ASiC-AIP	Associated Signature Container (ASiC)- Archival Information Package
ATS	ArchiveTimeStamp
AUG	Augmentation
CA	Certification Authority
CAB	Conformity Assessment Body
CLI (cli)	Command Line Interface
CRL	Certificate Revocation List
DMS	Data Management System
DSS	Digital Signature Services
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
et. seq.	et sequence
ECM	Enterprise Content Management
ER	Evidence Record
EU	European Union
EUMS	European Union Member State
GDPR	General Data Protection Regulation
IS-Policy	Information Security Policy (see e.g. [EN 319 401], chapter 6.3.)
IT	Information Technology
LXAIP	Logically XML formatted Archival Information Package
NC	Non-Conformity
OASIS	Organization for the Advancement of Structured Information Standards
OCSP	Online Certificate Status Protocol
OID	Object Identifier
OVR	Overall
PDS	Preservation of Digital Signature

Abbreviation	Keyword
PEP	Preservation Evidence Policy
PEPT	Preservation Evidence Policy Template
PGD	Preservation of General Data
PI	Potential for Improvement
PO	Preservation Object
POC	Preservation Object Container
PP	Preservation Profiles
PRP	Preservation Service Protocol
PS	Preservation Service
PSP	Preservation Service Provider
PSPS	Preservation Service Practice Statement
QES	Qualified Electronic Signature or qualified electronic seal
QTSP	Qualified Trust Service Provider
(Q)TPS	TSP or QTSP
QPSP	Qualified Preservation Service Provider
(Q)PSP	PSP or QPSP
R	Recommendation
SA	Subscriber Agreement
SSL	Secure Sockets Layer
SubDO	Submission Data Object
SVP	Signature Validation Policy
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
TSA	Time-Stamping Authority
TSP	Trust Service Provider
TS-Policy	Trust Service Policy
TSPS	Trust Service Practice Statement (see e.g. [EN 319 401], chapter 6.1.)
UTC	Coordinated Universal Time
WOS	Without Storage

Abbreviation	Keyword
WST	With Storage
WTS	With Temporary Storage
XAIP	XML formatted Archival Information Package
XML	Extensible Markup Language

Table 4: Keywords and Abbreviations

9 Bibliography

[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_EN_319_102-1] ETSI EN 319 102-1: *Electronic Signatures and Infrastructures (ESI); Procedures for Creation and Validation of AdES Digital Signatures; Part 1: Creation and Validation*, V1.3.1 (2021-11), https://www.etsi.org/deliver/etsi_en/319100_319199/31910201/01.03.01_60/en_31910201v010301p.pdf

[EN319122-3] ETSI TS 119 122 – 3, Electronic Signatures and Infrastructers (ESI); CAdES digital signatures, Part 3: Incorporation of ERS mechanisms in CAdES, V1.1.1, (2017-01) and higher, see http://www.etsi.org/deliver/etsi_ts/119100_119199/11912203/01.01.01_60/ts_11912203v010101p.pdf

[ETSI_TS_119_312] ETSI TS 119 312: *Electronic Signatures and Infrastructures (ESI); Cryptographic Suites*, V1.4.3 (2023-08), https://www.etsi.org/deliver/etsi_ts/119300_119399/119312/01.04.03_60/ts_119312v010403p.pdf

[ETSI_TS_119_511] ETSI TS 119 511, *Electronic Signatures and Infrastructures (ESI); Policy and security requirements for trust service providers providing long-term preservation of digital signatures or general data using digital signature techniques*, V1.1.1, (2019-06), https://www.etsi.org/deliver/etsi_ts/119500_119599/119511/01.01.01_60/ts_119511v010101p.pdf

[ETSI_TS_119_512] ETSI TS 119 512: Electronic Signatures and Infrastructures (ESI); *Protocols for trust service providers providing long-term data preservation services*, V1.2.1 (2023),

https://www.etsi.org/deliver/etsi_ts/119500_119599/119512/01.02.01_60/ts_119512v010201p.pdf

[OASIS-DSS] OASIS Standard: *Digital Signature Service Core Protocols, Elements, and Bindings, Version 1.0,* see http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf

[RFC4998] Gondrom, T., Brandner, R., Pordesch, u.: IETF RFC 4998 – Evidence Record Syntax (ERS), see http://www.ietf.org/rfc/rfc4998.txt

[TR-ESOR-E] BSI TR 03125-E: *Concretisation of the Interfaces on the Basis of the eCard-API-Framework: Annex TR-ESOR-E,* V1.3 and later versions, see https://www.bsi.bund.de/EN/tr-esor

[TR-ESOR-F] BSI TR 03125-F: *Preservation of Evidence of Cryptographically Signed Documents: Annex TR-ESOR-F Formats,* V1.3 and later versions, see https://www.bsi.bund.de/EN/tr-esor_XAIP

[TR-ESOR-VR] BSI TR 03125-VR: *Preservation of Evidence of Cryptographically Signed Documents: Annex TR-ESOR-VR: Verification Reports for Selected Data Structures*, V13 and later versions, see https://www.bsi.bund.de/EN/tr-esor

[TR-ESOR-ERS] BSI TR 03125: *Preservation of Evidential Value of Cryptographically Signed Documents*: *Annex TR-ESOR-ERS, Evidence Record pursuant to RFC4998 and RFC6283, V1.3 and later, see https://www.bsi.bund.de/EN/tr-esor*

[VDG] Vertrauensdienstegesetz – *VDG, Artikel 1 des Gesetzes zur Durchführung der Verordnung (EU) Nr. 910/2014 des Europäischen*

Parlaments und des Rates vom 23. Juli 2014 über elektronische Identifizierung und Vertrauensdienste für elektronische Transaktionen im Binnenmarkt und zur Aufhebung der Richtlinie 1999/93/EG (eIDAS-Durchführungsgesetz), Bundesgesetzblatt Jahrgang 2017 Teil I Nr. 52, ausgegeben zu Bonn am 28. Juli 2017