Bundesamt
für Sicherheit in der
Informationstechnik

BSI Technical Guideline 03125
Preservation of Evidence of Cryptographically Signed Documents

Evidence Record Verify Tool

Designation	ErVerify Tool
Abbreviation	BSI EVT
Version	1.3.5
Datum	16.02.2024

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

1. Introduction

1.1 Overview

Algorithms used for signatures compliant to [ETSI_EN_319_102-1] are applicable to create eIDAS qualified electronic signatures [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.

This tool aims at verifying the technical evidence records (ERS – Evidence Record Syntax) according to [RFC4998] associated with an XAIP, i.e. XML-formatted Archival Information Package, via a unique Archive Object ID (AOID). To achieve this, the `ERVerify Tool` 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 component to be configured separately.

1.2 Purpose

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

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.

This software is underlying the rules of the following license: Apache License Version 2.0, January 2004.

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

3.2 Command Line Application

After unpacking the distribution ZIP file, 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 cli/bin.
In case of Linux, you may want to make the script executable by calling `chmod u+x checktool`.
Before using the application, you have to create a valid configuration. See the following chapter for further details

3.3 Standalone Web Service

Install the command line application. To start the service, configure the application and call `checktool.sh -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 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 following chapter. 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:

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

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

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`
  - `TR-ESOR`
  - `Basis-ERS`

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 RFC 6283, section 4.2.2, point 6
    - `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.
  - `requireQualifiedTimestamps` (optional): 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 chose 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`
  - `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`

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`[1] elements. This may be necessary because web service access might disregard the prefixes used in a given 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.

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

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.

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, within a XAIP or within a CMS signature. Evidence records within a XAIP or CMS structure are only recognized if they are embedded correctly as specified in TR-ESOR 1.3 or CAdES, 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 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 and qualified timestamps)
- Any further profiles specified by administrator in the configuration.

4.3 Calling the Command Line Application

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

usage: java -jar ErVerifyTool-cli*.jar	
-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)
-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 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)
-server	start as web service (optional, ignores all other parameters except -conf and -port)

If the data parameter is a file containing a XAIP, then the evidence records embedded in that XAIP are checked as well. All other formats given as `data` parameter are handled as binary protected content and are not interpreted in any way.

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)

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

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

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

To start the stand-alone web service, call:

```
checktool -conf <FILE> -server -port <PORT>
```

By default, the command line application will create a log file named erVerifyTool.log in the working directory. To change logging behavior, set the system variable `log4j.configuration` to point to your custom Log4J2 configuration. 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
XAIP which may contain embedded evidence records as XML	/VerifyRequest/InputDocuments/Document/InlineXML/ XAIP
XAIP which may contain embedded evidence records as encoded XML	/VerifyRequest/InputDocuments/Document/Base64XML
CMS signature with embedded evidence records	/VerifyRequest/SignatureObject/Base64Signature

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

The evidence data for a XAIP can only be checked if the XML structure including any namespace prefixes can be properly reconstructed. When embedding a XAIP into the request as `InlineXML`, 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 XAIP into the request as additional namespaces might be incorrectly added into the canonicalized elements during hash value checks.

Providing a 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:
`{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:verificationre-port:reportdetail:noDetails`
- `urn:oasis:names:tc:dss-x:1.0:profiles:verificationre-port: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. The optional input `VerifyUnderSignaturePolicy` is currently not supported.

5. 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 time-stamps, 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/Con-figuredObjects` section of your configuration:

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

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

5.1 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.

5.2 Wrting 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`.

5.2.1 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 is given 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 the 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

5.2.2 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.

5.2.3 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).`

5.2.4 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 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>`.

5.2.5 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.

6. 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 API documentation 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.java
- EvidenceRecordValidator.java
- ArchiveTimeStampSequenceValidator
- ArchiveTimeStampValidator.java
- DummyTimeStampValidator.java
- 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.default_impl.ECardTim
eStampValidator
   </className>
   <targetType>org.bouncycastle.tsp.TimeStampToken</targetType>
</Validator>
```

Both validators `ECardTimeStampValidator` and `BasisErsECardTimeStampValidator` have been tested with Governikus Suite as service provider.

7. 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
CRL	Certificate Revocation List
DMS	Data Management System
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
OCSP	Online Certificate Status Protocol
OID	Object Identifier
OVR	Overall
PDS	Preservation of Digital Signature
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

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

Table 1: Keywords and Abbreviations

8. 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_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*, V1.1.1 (2016-05), https://www.etsi.org/deliver/etsi_en/319100_319199/31910201/01.01.01_60/en_31910201v010101p.pdf

[ETSI_TS_119_312] ETSI TS 119 312: *Electronic Signatures and Infrastructures (ESI); Cryptographic Suites*, V1.3.1 (2019-02), https://www.etsi.org/deliver/etsi_ts/119300_119399/119312/01.03.01_60/ts_119312v010301p.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*, V1.1.2 (2020), https://www.etsi.org/deliver/etsi_ts/119500_119599/119512/01.01.02_60/ts_119512v010102p.pdf [OASIS-DSS] OASIS Standard: *Digital Signature Service Core Protocols, Elements, and Bindings, Version 1.0,* siehe unter 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), siehe unter 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,* V1.2.1 and later versions

[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

[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