Federal Office
for Information Security

Technical Guideline TR-03129-3

Protocols for the Management of Certificates and CRLs in Public-Key-Infrastructures (PKIs)

Part 3: Electronic Identity (eID) documents based on Extended Access Control (EAC)

Version 1.40

Document history

Version	Date	Description
1.30	2016	Initial version. Split from TR-03129 v1.2
1.40	2022	Revised version

Federal Office for Information Security
Post Box 20 03 63
D-53133 Bonn

E-Mail: eid@bsi.bund.de
Internet: https://www.bsi.bund.de

Table of Contents

1. Introduction

This technical guideline specifies PKI-related communication protocols for security mechanisms in the context of electronic identity documents based on Extended Access Control (EAC). This document is part of the BSI TR-03129 series and depends on the [TR-03129-1] base document that describes the basic communication protocols, security mechanisms, web service messages, etc.

For PKI-related communication protocols for security mechanisms for machine readable travel documents with regard to document checks for public and official authorities refer to BSI [TR-03129-2].

Requirements for the CVCA-eID public key infrastructure (CVCA-eID PKI) and its participants are given in the certificate policy in [CP-eID].

1.1 Security Mechanisms

Data stored on electronic identity (eID) documents has to be protected against manipulation and unauthorized access by Passive Authentication (PA) and Terminal Authentication (TA), as part of Extended Access Control (EAC), respectively. Restricted Identification (RI) is a mechanism, that allows the usage of an eID document over networks in a privacy protecting way by providing "sector-specific" identifiers. These can be used to identify the eID document within each sector, instead of relying on globally unique identifiers like serial numbers or chip-individual public keys.

1.1.1 Supporting Lists

The security mechanisms are supported by three types of lists used for different purposes:

- **Master Lists:** To support the secure distribution of CSCA certificates, Master Lists according to [ICAO 9303] have been introduced. Master Lists are signed lists of (nationally) trusted CSCA certificates. Thus, the terminals only have to store the public key required for the verification of such a Master List securely.
- **Defect Lists:** A defect is defined as a production error affecting a large number of documents. The withdrawal of already issued documents is impractical or even impossible if the detected defect is contained in foreign documents. Defect Lists are signed lists to handle such defects. Defects are identified by the Document Signer Certificate(s) used to produce defect documents. Defect Lists are thus errata that not only inform about defects but also provides corrigenda to fix the error where possible.
- **Block Lists:** Block Lists are signed lists containing the sector-specific identifiers of revoked documents. As those identifiers are sector specific, a separate Block List is produced for every sector.

1.1.2 Passive Authentication

Passive Authentication is a mechanism standardized in [ICAO 9303], Part 11, Section 5.1 and [TR-03110-1], Sec. 1.1. Passive Authentication uses a digital signature to authenticate data stored on the eID documents. This signature is generated by a Document Signer (DS) in the personalization phase of the eID documents . The Document Signer in turn is certified by the Country Signing CA (CSCA) of the issuing country.

If a terminal has to read data from a eID document, the terminal must perform Passive Authentication to verify that the data is authentic and has not been manipulated. For Passive Authentication the terminal must have access to the corresponding Document Signer certificate and Country Signing CA certificate. While the Document Signer certificate can usually easily be obtained from the eID document itself, the CSCA certificate must be securely distributed to and stored on the terminal.

1.1.3 Terminal Authentication

Terminal Authentication is a mechanism standardized in [TR-03110-1] (TAv1) and [TR-03110-2] (TAv2). Terminal Authentication is based on a challenge-response protocol using digital signatures and card-verifiable (CV) certificates both to be validated by the eID document. After verification of the terminal certificate the eID document calculates the rights assigned to the terminal and grants the terminal access to the corresponding data stored on the eID document. The terminal certificate is issued by the Document Verifier (DV) the terminal is associated with. The Document Verifier in turn is certified by (at least) the national Country Verifying CA (CVCA) but it may also be certified by foreign CVCAs.

If a terminal has to read data protected by EAC from an eID document, the terminal must perform Terminal Authentication to prove that the terminal is authorized to access this data. For Terminal Authentication the terminal must present the eID document a certificate chain that starts at a certificate verifiable by the eID document using a trusted public key of the CVCA of the issuing country stored on the chip and ends at the terminal certificate for which the terminal possesses the corresponding private key.

1.1.4 Restricted Identification

Restricted Identification is a mechanism standardized in [TR-03110-2], Section 3.6. The mechanism provides sector-specific identifiers for the chip. The identifier calculated by Restricted Identification has the following properties:

- Within each sector the sector-specific identifier of every eID document's chip is unique.
- Across any two sectors, it is computationally impossible to link the two sector-specific identifier of any eID documents chip.

Restricted Identification can be used only after Terminal Authentication (and Chip Authentication) has been performed successfully between the terminal and the chip of a eID document. The calculation of the sector-specific identifier is based on a special public key of the sector, which is given by the terminal to the chip of the eID document. The chip proves the authenticity of this public key using the CV certificate of the terminal used within terminal authentication. By the key and certificate management it is assured that only the authorised owner of the key pair can use an eID document to calculate the sector-specific identifier of the chip of that eID document.

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119]. The key word “CONDITIONAL” is to be interpreted as follows:

CONDITIONAL: The usage of an item is dependent on the usage of other items. It is therefore further qualified under which conditions the item is REQUIRED or RECOMMENDED.

When used in tables (profiles), the key words are abbreviated as shown in Table 1.

Key word	Equivalent	Abbrev.
MUST / SHALL	REQUIRED	m
MUST NOT / SHALL NOT	–	x
SHOULD	RECOMMENDED	r
MAY	OPTIONAL	o
–	CONDITIONAL	c

Table 1: Key words

2. Implementation as Web Services

The implementation of communication protocols shall be realized as web services using SOAP-messages, which should be in conformance with the attached WSDL files.

2.1 Message format and common parameters and return codes

This guideline is part of the TR-03129 series.

All web service messages are specified according to the message format defined in [TR-03129-1], section 2. **Unless stated otherwise,** all common parameters and return codes listed in [TR-03129-1], section 2.2.1 and 2.2.2 are also valid throughout this document.

3. Protocols for CSCA Certificate Distribution - Passive Authentication

3.1 Specific Parameters and Return Codes

The following parameters and return codes are used frequently in this section in addition to the common parameters and return codes defined in [TR-03129-1].

3.1.1 Parameters

- `masterList`

  This parameter contains the Master List according to [ICAO 9303] part 12. This parameter is REQUIRED if a Master List will be sent with the message. It MUST be missing if no Master List will be sent with the message.
- `defectList`

  This parameter contains the Defect List according to Appendix A. This parameter is REQUIRED if a Defect List will be sent with the message. It MUST be missing if no Defect List will be sent with the message.

3.1.2 Return Codes

- `ok_list_available`

  The request has been processed successfully. The input or output parameter contains the requested list.
- `failure_list_not_available`

  The corresponding message has been processed correctly but the requested list is not available.

3.2 Distribution of MasterLists / CSCA Certificates

It is RECOMMENDED to distribute the CSCA certificates using MasterLists since they are the commonly used instrument. A MasterList is published by the NPKD (National Public Key Directory) to distribute all trustworthy CSCA certificates to the subordinated organisations. The MasterLists are distributed via the CVCA.

The following messages SHALL be used for the distribution of Master Lists [ICAO 9303].

3.2.1 GetMasterList

This message is sent by a terminal to its DV or by the DV to the CVCA in order to get one or more (signed) lists of trusted CSCA certificates.

If the DV or the CVCA processes this message asynchronously, it MUST must respond using the `SendMasterList` message.

Input parameters:

- `callbackIndicator` **(REQUIRED)**
- `messageID` **(CONDITIONAL)**
- `responseURL` **(DEPRECATED), (OPTIONAL)**

Output parameters:

- `masterList` **(CONDITIONAL)**
- `returnCode` **(REQUIRED)**

  The following return codes are possible:

  - `failure_incorrect_request`
  - `failure_internal_error`
  - `failure_list_not_available`
  - `failure_other_error`
  - `failure_synchronous_processing_not_possible`
  - `failure_syntax`
  - `ok_list_available`
  - `ok_reception_ack` **(DEPRECATED)**
  - `ok_syntax`
- `returnCodeMessage` **(OPTIONAL)**

3.2.2 SendMasterList

If the CVCA or the DV processes the message `GetMasterList` asynchronously, it uses a response message `SendMasterList` to communicate the result of its processing.

This message can also be used to notify registered entities about the availability of a new master list. In this case the `messageID` MUST be omitted.

This message is responded by a return code only.

Input parameters:

- `messageID` **(CONDITIONAL)**
- `statusInfo` **(REQUIRED)**

  The following status codes are possible:

  - `failure_incorrect_request`
  - `failure_internal_error`
  - `failure_list_not_available`
  - `failure_other_error`
  - `failure_syntax`
  - `ok_list_available`
- `statusInfoMessage` **(OPTIONAL)**
- `masterList` **(CONDITIONAL)**

Output parameters:

- `returnCode` **(REQUIRED)**

  The following return codes are possible:

  - `failure_internal_error`
  - `failure_messageID_unknown`
  - `failure_other_error`
  - `failure_syntax`
  - `ok_received_correctly`
- `returnCodeMessage` **(OPTIONAL)**

3.2.3 GetDefectList

This message is sent by a terminal to its DV or by the DV to the CVCA in order to get one or more (signed) lists of defect or revoked CSCA and DS certificates.

If the DV or the CVCA processes this message asynchronously, it SHALL respond using the `SendDefectList` message.

Input parameters:

- `callbackIndicator` **(REQUIRED)**
- `messageID` **(CONDITIONAL)**
- `responseURL` **(DEPRECATED), (OPTIONAL)**

Output parameters:

- `defectList` **(CONDITIONAL)**
- `returnCode` **(REQUIRED)**

  The following return codes are possible:

  - `failure_incorrect_request`
  - `failure_internal_error`
  - `failure_list_not_available`
  - `failure_other_error`
  - `failure_synchronous_processing_not_possible`
  - `failure_syntax`
  - `ok_list_available`
  - `ok_reception_ack` **(DEPRECATED)**
  - `ok_syntax`
- `returnCodeMessage` **(OPTIONAL)**

3.2.4 SendDefectList

If the CVCA or the DV processes the message `GetDefectList` asynchronously, it uses a callback message `SendDefectList` to communicate the result of its processing.

This message can also be used to notify registered entities about the availability of a new defect list. In this case the `messageID` SHALL be omitted.

This message is responded by a return code only.

Input parameters:

- `messageID` **(CONDITIONAL)**
- `statusInfo` **(REQUIRED)**

  The following status codes are possible:
  - `failure_incorrect_request`
  - `failure_internal_error`
  - `failure_list_not_available`
  - `failure_other_error`
  - `failure_syntax`
  - `ok_list_available`
- `statusInfoMessage` **(OPTIONAL)**
- `defectList` **(CONDITIONAL)**

Output parameters:

- `returnCode` **(REQUIRED)**

  The following return codes are possible:
  - `failure_internal_error`
  - `failure_messageID_unknown`
  - `failure_other_error`
  - `failure_syntax`
  - `ok_received_correctly`
- `returnCodeMessage` **(OPTIONAL)**

3.3 Implementation as Web Services

It is RECOMMENDED that the protocol given in this chapter is realized as a Web Service using the SOAP messages described in the attached WSDL specifications.

3.3.1 Services Implemented by the CVCA

A CVCA MUST implement a Web Service supporting the following messages:

- `GetMasterList`
- `GetDefectList`

3.3.2 Services Implemented by DVs

For the communication with its associated inspection systems a DV MUST implement a Web Service supporting the following messages:

- `GetMasterList`
- `GetDefectList`

If a DV supports callback messages sent by the CVCA, the Web Service of the DV MUST in addition also support the following (callback) messages:

- `SendMasterList`
- `SendDefectList`

3.3.3 Services Implemented by Terminals

A terminal has to implement the corresponding Web Services only if they support callback messages sent by the DV. In this case they have to implement Web Services supporting the following (callback) messages:

- `SendMasterList`
- `SendDefectList`

4. Protocols for CVCA and DV certificate distribution -Terminal Authentication

This section provides sector-specific protocols for the distribution of CV certificates and related data between CVCAs, DVs, and terminals in addition to [TR-03129-1].

4.1 Specific Parameters and Return Codes

The following parameters and return codes are used frequently in this section in addition to the parameters and return codes defined in Ch. 2.2 of [TR-03129-1].

4.1.1 Parameters

- `certReq`

  This parameter contains the certificate request. It MUST be constructed according to Appendix C.2 of [TR-03110-3]. The encoding MUST follow the specifications in Appendix D of [TR-03110-3].

- `noPollBefore`

  This parameter gives the earliest possible UNIX timestamp when a certificate may be available for polling.

- `cmsContainer`

  This parameter contains the CMS container of `ContentType SignedData` according to [RFC 5652]. The CMS container is used to transmit TLS server certificates and Certificate Signing Requests. A dedicated certificate, so-called Request Signer Certificate (RSC), is used to sign the CMS container. The specification of the CMS container is defined in [TR-03129-Annex]. The specification of the Request Signer Certificate is defined in [CP-eID].

4.1.2 Return Codes

- `ok_entanglement_successful`

  This code should be returned, if the TLS-certificate has been entangled with the CV certificate correctly.

- `failure_signature`

  The verification of the signature of the link certificate failed.

- `failure_domain_parameters`

  The domain parameters contained in the request do not match the domain parameters of the corresponding CVCA certificate.

- `failure_request_not_accepted`

  The message has been processed correctly but the request has not been accepted. In this case further procedures must be determined by organisational measures.

- `failure_certificate_holder_unknown`

  The message has been processed correctly but the certificate holder reference in the request does not contain a holder mnemonic known by the certification authority.

- `failure_certification_authority_holder_unknown`

The message has been processed correctly but the certification authority indicated in the inner CAR of the request is unknown.

- `failure_certificate_holder_inactive`

  The message has been processed correctly but the certificate holder is currently or permanently marked inactive.

- `failure_cert_not_available`

  This code should be returned, if the BerCA has not yet issued a certificate corresponding to the pending Certificate Signing Request.

- `failure_entanglement_error`

  This code should be returned, if the TLS-certificate cannot be entangled with the CV certificate correctly.

- `failure_not_authorized`

  The message has been processed correctly but the requesting entity is not authorized to request a certificate for this certificate holder.

- `failure_certificate_holder_reference_in_use`

  The message has been processed correctly but the certificate holder reference in the request has already been used in a recent certificate request.

- `failure_inner_signature`

  The verification of the inner signature of the actual certificate request according to [TR-03110-3] failed or the verification of the signature of the transported certificates in the CMS container failed.

- `failure_outer_signature`

  The verification of the outer signature of the actual certificate request according to [TR-03110-3] failed or the verification of the signature of the CMS container failed..

- `warning_noQWAC`

  This code should be returned, if the TLS server certificate is not a qualified website authentication certificate.

  Application note: This return code is only informational and should not abort any process.

4.2 Distribution of CV Certificates, TLS-Certificates and Request-Signer Certificates

The following messages `RequestCertificate`, `GetCACertificates`, `GetCertificate` and `SendCertificates` are used for the management of CV certificates described in [TR-03110-3] at a national level in addition to the default messages defined [TR-03129-1]:

4.2.1 RequestCertificate

This message is used by a terminal to request the generation of a new certificate for one of its keys from the DV. If the parent DV processes this message asynchronously, it MUST respond using the `SendCertificates` message.

The message `RequestCertificate` is defined as described in [TR-03129-1], except for the following **additional or refined** parameters.

Input parameters:

- `certReq` **(REQUIRED)**

  This parameter contains the certificate request. It MUST be constructed according to Appendix C.2 of [TR-03110-3]. The encoding MUST follow the specifications in Appendix D of [TR-03110-3].

Output parameters:

- `certificateSeq` **(CONDITIONAL)**
- `returnCode` **(REQUIRED)**

  All return codes defined for `RequestCertificate` in [TR-03129-1] and the following **additional** return codes are possible:

  - `failure_certificate_holder_inactive`
  - `failure_certificate_holder_reference_in_use`
  - `failure_certificate_holder_unknown`
  - `failure_certification_authority_holder_unknown`
  - `failure_domain_parameters`
  - `failure_expired`
  - `failure_inner_signature`
  - `failure_not_authorized`
  - `failure_outer_signature`
  - `failure_request_not_accepted`
  - `ok_reception_ack` **(DEPRECATED)**

4.2.2 GetCACertificates (Deprecated)

Renamed to GetCertificates. See below.

4.2.3 GetCertificates

This message is sent by a DV or by a terminal to a CVCA or to a DV, respectively, in order to get all relevant CA certificates of the national CVCA or foreign CVCAs. In this sense, a CA certificate is regarded as relevant if it is still valid and if it is needed for the verification of a (valid) certificate of the DV (generated by the CVCA).

The message requests the DV or CVCA to return a chain of certificates including all valid link certificates, beginning with the first Root-CA certificate.

If the parent DV or CVCA processes this message asynchronously, it SHALL respond using the `SendCertificates` message.

The message `GetCertificates` is defined as described in [TR-03129-1].

Deprecation Note 1: *This message* `GetCertificates` *replaces the deprecated message* `GetCACertificates` *which MAY be implemented for a transitional period to provide backward compatibility. They will be removed in a future version of this document.*

4.2.4 SendCertificates

If a CVCA or DV processes one of the messages `RequestCertificate` or `GetCertificates` asynchronously, it uses a response message `SendCertificates` to communicate the result of its processing. It sends the response message always to that URL which is determined from the TLS-certificate of the sender of the request-message.

This message can also be used to notify registered entities about the availability of new certificates. In this case the `messageID` MUST be omitted.

This message itself MUST always be processed synchronously by its receiver.

The message `SendCertificates` is defined as described in [TR-03129-1], except for the following **additional or refined** parameters.

Input parameters:

- `statusInfo` **(REQUIRED)**

  All status codes defined for `SendCertificates` in [TR-03129-1] and the following **additional** status codes are possible:

  - `failure_domain_parameters`
  - `failure_expired`
  - `failure_inner_signature`
  - `failure_outer_signature`
  - `failure_request_not_accepted`

4.2.5 GetCertificateDescription

This message is only used by an eID-Server that participates in the Online-Authentication based on Extended Access Control Version 2 (EAC2) between an eService and an eID document. Hereby, the eID-Server assumes the role of the remote terminal.

This message is sent by an eID-Server to its DV (which is called BerCA in that case) in order to get the `CertificateDescription` of a specific terminal CV certificate.

To enable the BerCA to return the corresponding `CertificateDescription`, the hash value `CertificateDescriptionHash` given in the respective terminal CV certificate must be provided by the eID-Server.

Input parameters:

- `hash` **(REQUIRED)**

  This parameter MUST include the hash value of the `CertificateDescription` object as included in the `CertificateDescriptionHash` extension of the remote terminal CV certificate.

Output parameters:

- `certificateDescription` **(CONDITIONAL)**

  This parameter contains the DER-encoded ASN.1 structure of the `certificateDescription` object (as described in [TR-03110-4]) for which the hash value is included into the `CertificateDescriptionHash` extension of the remote terminal certificate.

- `returnCode` **(REQUIRED)**

The following return codes are possible:

- `failure_incorrect_request`
- `failure_internal_error`
- `failure_other_error`
- `failure_syntax`
- `ok_received_correctly`

- `returnCodeMessage` **(OPTIONAL)**

4.2.6 SendeIDServerCerts

This message is only used by an eID-Server that participates in the Online-Authentication based on Extended Access Control Version 2 (EAC2) between an eService and an eID document. Hereby, the eID-Server assumes the role of the remote terminal.

This message is sent by an eID-Server to its DV (which is called BerCA in that case) in order to submit the X.509 (TLS) certificates for calculating the hash value of the TLS certificates placed in the set `commCertificates` according to [TR-03110-4], 2.2.6.

The certificates shall be shipped within a CMS container of `ContentType SignedData`. To sign the CMS container, a valid RSC shall be used. The specification of the CMS container and the application-specific OID, defining the encapsulated content, is given in [TR-03129-Annex].

Input parameters:

- `cmsContainer` **(REQUIRED)**

Output parameters:

- `returnCode` **(REQUIRED)**

  The following return codes are possible:

  - `failure_eContentType`
  - `failure_entanglement_error`
  - `failure_expired`
  - `failure_inner_signature`
  - `failure_internal_error`
  - `failure_other_error`
  - `failure_outer_signature`
  - `failure_syntax`
  - `ok_entanglement_successful`
  - `warning_noQWAC`

- `returnCodeMessage` **(OPTIONAL)**

4.2.7 RequestPKICommunicationCert

This message is only used by an eID-Server that participates in the Online-Authentication based on Extended Access Control Version 2 (EAC2) between an eService and an eID document. Hereby, the eID-Server assumes the role of the remote terminal.

This message is sent by an eID-Server to its DV (which is called BerCA in that case) in order to submit the PKCS#10 Certificate Signing Request for renewing the TLS client certificate regarding PKI communication. This message shall be processed synchronously and works in conjunction with the message `GetPKICommunicationCert`.

The request shall be shipped within a CMS container of `ContentType SignedData`. To sign the CMS container, a valid RSC shall be used. The specification of the CMS container and the application-specific OID, defining the encapsulated content, is given in [TR-03129-Annex].

After the BerCA has successfully validated the CMS container and the request itself, the BerCA issues a new TLS client certificate regarding PKI communication. The transfer of the TLS client certificate back to the eID-Server is based on a polling mechanism. For the eID-Server to receive the TLS client certificate issued by the BerCA, the message `GetPKICommunicationCert` must be called subsequently. The BerCa should send the earliest possible timestamp when the next certificate may be available using `noPollBefore`.

Input parameters:

- `messageID` **(REQUIRED)**
- `cmsContainer` **(REQUIRED)**

Output parameters:

- `noPollBefore` **(OPTIONAL)**
- `returnCode` **(REQUIRED)**

  The following return codes are possible:
  - `failure_eContentType`
  - `failure_expired`
  - `failure_incorrect_request`
  - `failure_inner_signature`
  - `failure_internal_error`
  - `failure_other_error`
  - `failure_outer_signature`
  - `failure_request_not_accepted`
  - `failure_syntax`
  - `ok_received_correctly`
- `returnCodeMessage` **(OPTIONAL)**

4.2.8 GetPKICommunicationCert

This message is only used by an eID-Server that participates in the Online-Authentication based on Extended Access Control Version 2 (EAC2) between an eService and an eID document. Hereby, the eID-Server assumes the role of the remote terminal.

This message is sent by an eID-Server to its DV (which is called BerCA in that case) in order to poll for the TLS client certificate issued by the BerCA. This message shall be processed synchronously and works in conjunction with the message `RequestPKICommunicationCert`.

After the eID-Server has called the message `RequestPKICommunicationCert`, the message `GetPKICommunicationCert` must be called with the same `messageID` to retrieve the issued TLS client certificate. If the BerCA has not yet fully processed the PKCS#10 Certificate Signing Request and therefore the new TLS client certificate has not yet been issued, the message `GetPKICommunicationCert` must be called again by the eID-Server. The BerCa should send the earliest possible timestamp when the next certificate may be available using `noPollBefore.` A successful polling request using the message `GetPKICommunicationCert` results in the return of the corresponding TLS client certificate.

Input parameters:

- `messageID` **(REQUIRED)**

Output parameters:

- `certificateSeq` **(CONDITIONAL)**
- `noPollBefore` **(OPTIONAL)**
- `returnCode` **(REQUIRED)**

  The following return codes are possible:

  - `failure_cert_not_available`
  - `failure_internal_error`
  - `failure_messageID_unknown`
  - `failure_other_error`
  - `failure_syntax`
  - `ok_cert_available`
- `returnCodeMessage` **(OPTIONAL)**

4.2.9 SendRSCCert

This message is only used by an eID-Server that participates in the Online-Authentication based on Extended Access Control Version 2 (EAC2) between an eService and an eID document. Hereby, the eID-Server assumes the role of the remote terminal.

This message is sent by an eID-Server to its DV (which is called BerCA in that case) in order to submit the new Request Signer Certificate intended to replace the previously submitted RSC.

The message `SendRSCCert` is defined as described in [TR-03129-1],

4.3 Implementation as Web Services

It is RECOMMENDED that the protocol given in this chapter is realised as a Web Service using the SOAP messages described in the attached WSDL specifications.

4.3.1 Services Implemented by the CVCA/SPOCs

A CVCA/SPOC has to implement a Web Service supporting the following (callback) messages:

- `RequestCertificate`
- `GetCertificates`

4.3.2 Services Implemented by DVs

For the communication with its associated inspection systems a DV has to implement a Web Service supporting the following messages:

- `RequestCertificate`
- `GetCertificates`

If a DV supports callback messages sent by the CVCA/SPOC, the Web Service of the DV MUST in addition also support the following (callback) messages:

- `SendCertificates`

The following message needs only to be implemented, if the DV acts as a BerCA and issues authorization certificates to service providers for the use of the online authentication functionality:

- `GetCertificateDescription`
- `SendeIDServerCerts`
- `RequestPKICommunicationCert`
- `GetPKICommunicationCert`
- `SendRSCCert`

4.3.3 Services Implemented by Terminals

A terminal has to implement the following Web Service only, if they support callback messages sent by their DV. In this case they have to implement the Web Services supporting the following (callback) messages:

- `SendCertificates`

5. Protocols for Restricted Identification

This section contains the description of the messages related to Restricted Identification.

5.1 Specific Parameters and Return Codes

The following input and output parameters are used frequently in this section.

5.1.1 Parameters

- `deltaIndicator`

  This parameter indicates, if the caller wants to download the complete block list or only a delta list. In the first case the parameter is set to `complete_list`, in the second case it is set to `delta_list`.

- `deltaBase`

  This parameter contains the list identifier (`listID`, cf. Appendix B.1) of the block list currently available to the caller if the `deltaIndicator` parameter is set to `delta_list`. Then the delta request will provide the delta items (added, removed) in respect to this version.

- `deltaListAddedItems`

  If a delta list is transferred (`deltaIndicator = delta_list`) this parameter contains the list of items added to the block list since the version given in the `deltaBase` parameter. The list is empty if no item was added between the given `deltaBase` and the current block list. This parameter is REQUIRED if a delta list will be sent with the response. It MUST be missing if no delta list will be sent with the response.

- `deltaListRemovedItems`

  If a delta list is transferred (`deltaIndicator = delta_list`) this parameter contains the list of items removed since the version given in the `deltaBase` parameter. The list is empty if no item was removed between the given `deltaBase` and the current block list. This parameter is REQUIRED if a delta list will be sent with the response. It MUST be missing if no delta list will be sent with the response.

- `completeListURL`

  This parameter contains the URL, where the caller can download the latest version of the complete block list. After downloading the complete list from the URL the caller SHOULD trigger a further delta list request. This is because the block list provider may provide the complete list only at a particular time e.g. only on a daily base. This parameter is REQUIRED if a complete list will be sent with the response. It MUST be missing if no complete list will be sent with the response.

If delta lists are generated, `deltaListAddedItems` and `deltaListRemovedItems` MUST NOT contain the same sector-specific identifiers.

5.1.2 Return Codes

- `ok_list_available`

  The message has been processed successfully. Either the output parameters `deltaListAddedItems` and `deltaListRemovedItems` contain the delta information in respect to the base version, or the output parameter `completeListURL` contains the URL where the caller can download the new list.

- `ok_no_update_needed`

The message has been processed successfully, but there is no update for the block list needed. The block list currently available by the caller (as indicated by the parameter `deltaBase`) is still up to date.

- `ok_complete_list`

  The caller has requested a delta list, but its available block list version (as indicated in the `deltaBase` input parameter) is too old to be updated by a delta list or the size of the delta list would exceed certain technically implied constraints. A complete list (i.e. URL) is provided instead.

- `failure_deltaBase_unknown`

  The identifier `deltaBase` is unknown by the receiver. In this case further procedures must be determined by organisational measures.

- `failure_sectorID_unknown`

  The sector identifier is unknown by the DV/MBS. In this case further procedures must be determined by organisational measures.

5.2 Distribution of Sector Public Keys

The following messages are used for the management of public keys required for Restricted Identification.

5.2.1 GetSectorPublicKey

This message is used by an authentication terminal to download its sector public key from the DV it is associated to. The message is also used by a DV to download the sector public key of the revocation sector from the Master Blocklisting Service (MBS).

Input parameters:

- `sectorID` **(REQUIRED)**

  This parameter contains the identifier of the requested sector public key. The identifier is contained in the terminal certificate issued by the DV and in the block list provided by the Master Blocklisting Service (MBS).

Output parameters:

- `sectorPK` **(CONDITIONAL)**

  This parameter will contain the sector public key as data object (cf. [TR-03110-3], Appendix B4.1), if the message has been processed successfully.

- `returnCode` **(REQUIRED)**
  - `failure_internal_error`
  - `failure_other_error`
  - `failure_sectorID_unknown`
  - `failure_syntax`
  - `ok_PK_available`

    The message has been processed successfully. The output parameter sectorPK contains the requested sector public key.

- `returnCodeMessage` **(OPTIONAL)**

5.3 Distribution of Block Lists

The following sections contain the description of the messages for the distribution of block lists (cf. Appendix B).

5.3.1 GetBlockList

This message is used by a DV or an authentication terminal to download the most recent block list from the MBS or DV, respectively.

The caller can indicate if it wishes to download the complete block list or only a delta list. If the caller requests the download of the complete list, the block list provider MUST provide a complete list. Block list providers MAY support delta list requests. If the caller requests a delta list but the block list provider does not support delta lists it MUST return a complete block list. An eID-Server supporting multiple sectors will receive BlockListDetails for all sectors, unless a specific sector is selected using the sectorID field

Input parameters:

- `callbackIndicator` **(REQUIRED)**
- `messageID` **(CONDITIONAL)**
- `responseURL` **(DEPRECATED), (OPTIONAL)**
- `deltaIndicator` **(REQUIRED)**
- `deltaBase` **(CONDITIONAL)**
- `sectorID` **(OPTIONAL)**

  This message is only used by an eID-Server that supports multiple eServices. The parameter `sectorID` indicates the eService's sector identifier for which the block list is requested.

Output parameters:

- `deltaListAddedItems` **(CONDITIONAL)**
- `deltaListRemovedItems` **(CONDITIONAL)**
- `completeListURL` **(CONDITIONAL)**
- `returnCode` **(REQUIRED)**

  The following return codes are possible:

  - `failure_deltaBase_unknown`
  - `failure_sectorID_unknown`
  - `failure_internal_error`
  - `failure_other_error`
  - `failure_synchronous_processing_not_possible`
  - `failure_syntax`
  - `ok_complete_list`
  - `ok_list_available`
  - `ok_no_update_needed`
  - `ok_reception_ack` **(DEPRECATED)**

  - `ok_syntax`
- `returnCodeMessage` **(OPTIONAL)**

5.3.2 SendBlockList

If the MBS or a DV processes the message `GetBlockList` asynchronously, it uses a response message `SendBlockList` to communicate the result of its processing. It sends the response message always to that URL which is determined from the TLS-certificate of the sender of the request-message.

This message can also be used to notify registered entities about the availability of a new block list. In this case the `messageID` MUST be omitted.

This message itself MUST always be processed synchronously by its receiver.

Input parameters:

- `messageID` **(CONDITIONAL)**
- `statusInfo` **(REQUIRED)**

  The following status codes are possible:

  - `failure_deltaBase_unknown`
  - `failure_sectorID_unknown`
  - `failure_internal_error`
  - `failure_other_error`
  - `failure_syntax`
  - `ok_complete_list`
  - `ok_list_available`
  - `ok_no_update_needed`
- `statusInfoMessage` **(OPTIONAL)**
- `deltaListAddedItems` **(CONDITIONAL)**
- `deltaListRemovedItems` **(CONDITIONAL)**
- `completeListURL` **(CONDITIONAL)**

Output parameters:

- `returnCode` **(REQUIRED)**

  The following return codes are possible:

  - `failure_internal_error`
  - `failure_messageID_unknown`
  - `failure_other_error`
  - `failure_syntax`
  - `ok_received_correctly`
- `returnCodeMessage` **(OPTIONAL)**

5.4 Implementation as Web Services

It is RECOMMENDED that the protocol given in this chapter is realized as a Web Service using the SOAP messages described in the attached WSDL specifications.

5.4.1 Services Implemented by the CVCA/MBS

The CVCA/MBS has to implement a Web Service supporting the following messages:

- `GetSectorPublicKey`
- `GetBlockList`

5.4.2 Services Implemented by DVs

For the communication with its associated authentication terminals a DV has to implement a Web Service supporting the following messages:

- `GetSectorPublicKey`
- `GetBlockList`

If a DV supports callback messages sent by the CVCA/MBS, the Web Service of the DV MUST in addition also support the following (callback) messages:

- `SendBlockList`

5.4.3 Services Implemented by Authentication Terminals

An authentication terminal has to implement the corresponding Web Service only, if it supports callback messages sent by its DV. In this case it has to implement a Web Service supporting the following (callback) messages:

- `SendBlockList`

A Defect Lists

Defect Lists are signed lists to handle production error affecting a large number of documents. Defects are identified by the Document Signer Certificate(s) used to produce defect documents. Defect Lists are thus errata that not only inform about defects but also provides corrigenda to fix the error where possible.

Note that it is possible that a document does not contain a defect although the respective Document Signer Certificate is on the Defect List.

The object identifiers used in this appendix are contained in the subtree of `bsi-de`:

```
bsi-de OBJECT IDENTIFIER ::= {
  itu-t(0) identified-organization(4) etsi(0)
  reserved(127) etsi-identified-organization(0) 7
}
```

A.1 Defect List Format

A Defect List SHALL be provided as BER encoded `SignedData` according Appendix C with content type `DefectList` identified by `id-DefectList`:

```
id-DefectList{
  bsi-de applications(3) MRTD(1) 5}
}

DefectList ::= SEQUENCE {
  version   INTEGER {v1(0)}
  hashAlg   OBJECT IDENTIFIER,
  defects   SET OF Defect
}

Defect ::= SEQUENCE{
  signerIdentifier SignerIdentifier,
  certificateHash  OCTET STRING OPTIONAL,
  knownDefects     SET OF KnownDefect
}

KnownDefect ::= SEQUENCE{
  defectType       OBJECT IDENTIFIER,
  parameters       ANY defined by defectType OPTIONAL
}
```

The data type `SignerIdentifier` is defined in [RFC 5652] as follows:

```
SignerIdentifier ::= CHOICE {
  issuerAndSerialNumber    IssuerAndSerialNumber,
  subjectKeyIdentifier [0] SubjectKeyIdentifier
}
```

A.1.1 Identification of Document Signer Certificates

There are two main options to reference the Document Signer Certificate in a Defect List, either by the distinguished name of the CSCA and the serial number of the Document Signer Certificate or by the Subject Key Identifier of the Document Signer Public Key. Those two options are augmented by a third option: If neither `issuerAndSerialNumber` nor the `subjectKeyIdentifier` uniquely identify the Document Signer (due to a defect in the certificate) the hash of the Document Signer Certificate MUST additionally be included. The hash function to be used is indicated in the Defect List.

A.2 Defect Categories

Three groups of document defect types are currently distinguished:

1. Authentication Defects
2. Application Defects
3. Document Defects

The following section specify the object identifier for defects in those categories and the parameters describing workarounds for the defect.

Table 2 specifies the content of three types of Defect Lists for different purposes:

- **Internal:** This defect list is used for internal purposes, in particular for border control. It MUST contain all known defects.
- **External:** This defect lists is used to inform other countries about defects in national documents in particular ePassports. It MUST NOT contain defects referring to foreign documents.
- **Commercial:** This defect list is used to inform private companies about defects in national documents. It SHOULD NOT contain defects referring to foreign documents.

Defect Type	Internal	External	Commercial
Authentication			
`id-CertRevoked`	m	m	m
`id-CertReplaced`	m	m	m
`id-ChipAuthKeyRevoked`	m	m	m
`id-ActiveAuthKeyRevoked`	m	m	m
Application			
`id-ePassportDGMalformed`	m	m	o
`id-SODInvalid`	m	m	o
`id-eIDDGMalformed`	m	o	r
`id-eIDIntegrity`	m	o	r
Document			
`id-CardSecurityMalformed`	m	o	m
`id-ChipSecurityMalformed`	m	o	x
`id-PowerDownReq`	m	o	m

Table 2: Defect Lists

A.2.1 Authentication Defects

This section describes defects related to Passive Authentication, Chip authentication, and Active Authentication. This category of defects is indicated by the following object identifier:

```
id-AuthDefect     ::= OBJECT IDENTIFIER{id-DefectList 1}
```

If a document is affected by an authentication defect (i.e. the defect is contained the sub-tree of `id-AuthDefect`) with unknown interpretation, the electronic part of the document MUST NOT be used.

A.2.1.1 Document Signer Certificate Revoked

The private key of the Document Signer is compromised (e.g. revoked by a CRL). The electronic part of the document MUST NOT be used.

Note that this defect is equivalent to an issued CRL containing the Document Signer. This type of defect can be used to revoke foreign Document Signers independent of the availability of a CRL issued by the respective CSCA.

This type of defect is indicated by the following object identifier. It provides a status code (`StatusCode`) as parameter.

```
id-CertRevoked  ::= OBJECT IDENTIFIER{id-AuthDefect 1}

StatusCode ::= ENUMERATED {
  noIndication(0),     -- no details given
  onHold(1),           -- revocation under investigation
  testing(2),          -- the certificate has been used for testing purposes
  revokedByIssuer(3),  -- the Issuer has revoked the certificate by CRL
  revokedDLS(4),       -- the Defect List Signer has revoked the certificate
  proprietary(32)      -- status codes >=32 can be used for internal purposes
}
```

A.2.1.2 Document Signer Certificate Malformed

The Document Signer Certificate cannot be decoded correctly using standardized mechanisms. A replacement certificate is provided.

Note that this certificate is not necessarily signed by the corresponding CSCA. Replacement Document Signer Certificates MAY be self-signed.

This type of defect is indicated by the following object identifier. It provides a replacement certificate (`Certificate`, cf. [RFC 5280]) as parameter.

```
id-CertReplaced ::= OBJECT IDENTIFIER{id-AuthDefect 2}
```

A.2.1.3 Chip Authentication Private Keys Compromised

The Chip Authentication Private Keys have been compromised (e.g. due to an error in the key generation algorithm).

Chip Authentication SHOULD be used, but the successful execution does neither guarantee that the document nor the transferred data is genuine.

This type of defect is indicated by the following object identifier. It does not provide any parameters:

```
id-ChipAuthKeyRevoked  ::= OBJECT IDENTIFIER{id-AuthDefect 3}
```

A.2.1.4 Active Authentication Private Keys Compromised

The Active Authentication Private Keys have been compromised (e.g. due to an error in the key generation algorithm).

Active Authentication SHOULD be used, but the successful execution does not guarantee that the document is genuine.

This type of defect is indicated by the following object identifier. It does not provide any parameters:

```
id-ActiveAuthKeyRevoked  ::= OBJECT IDENTIFIER{id-AuthDefect 4}
```

A.2.2 Personalisation Defects of the ePassport Application

The following sections describe defects related to the personalisation of the ePassport application. This category of defects is indicated by the following object identifier:

```
id-ePassportDefect ::= OBJECT IDENTIFIER{id-DefectList 2}
```

If a document is affected by an personalization defect of the ePassport application (i.e. the defect is contained the sub-tree of `id-ePassportDefect`) with unknown interpretation, the ePassport application MUST NOT be used.

A.2.2.1 Data Group Malformed

The indicated data groups might be incorrectly encoded. In this case the data group SHOULD be ignored and manual inspection is REQUIRED.

This type of defect is indicated by the following object identifier. It provides the malformed DGs (`MalformedDGs`) as parameter:

```
id-ePassportDGMalformed  ::= OBJECT IDENTIFIER{id-ePassportDefect 1}
MalformedDGs    ::= SET OF INTEGER -- DGs as integer
```

A.2.2.2 Document Security Object Malformed

The validation of the Document Security Object might fail (e.g. signature incorrect). In this case the electronic part of the document MUST NOT be used.

This type of defect is indicated by the following object identifier. It does not provide any parameters:

```
id-SODInvalid   ::= OBJECT IDENTIFIER{id-ePassportDefect 2}
```

A.2.3 Personalisation Defects of the eID Application

The following sections describe defects related to the personalisation of the eID application. This category of defects is indicated by the following object identifier:

```
id-eIDDefect ::= OBJECT IDENTIFIER{id-DefectList 3}
```

If a document is affected by an personalization defect of the eID application (i.e. the defect is contained the sub-tree of `id-eIDDefect`) with unknown interpretation, the eID application SHOULD NOT be used.

A.2.3.1 Data Group Malformed

The indicated data groups might be incorrectly encoded. In this case the data group SHOULD be ignored.

This type of defect is indicated by the following object identifier. It provides the malformed DGs (`MalformedDGs`) as parameter:

```
id-eIDDGMalformed  ::= OBJECT IDENTIFIER{id-eIDDefect 1}
```

A.2.3.2 Application Integrity Uncertain

The integrity of unsigned data groups is not guaranteed.

This type of defect is indicated by the following object identifier. It does not provide any parameter:

```
id-eIDIntegrity  ::= OBJECT IDENTIFIER{id-eIDDefect 2}
```

A.2.4 General Document Defects

The following sections describe defects related to the document in general. This category of defects is indicated by the following object identifier:

```
id-DocumentDefect ::= OBJECT IDENTIFIER{id-DefectList 4}
```

If a document is affected by a general defect (i.e. the defect is contained the sub-tree of `id-DocumentDefect`) with unknown interpretation, the electronic part of the document SHOULD NOT be used.

A.2.4.1 Card Security Object Malformed

The Card Security Object is incorrectly encoded. A corrected Card Security Object is provided that SHOULD be used instead.

This type of defect is indicated by the following object identifier. It provides a corrected Card Security Object (`id-SecurityObject`, cf. [TR-03110-3], Appendix A.1.2.2 and A.1.2.5) as parameter:

```
id-CardSecurityMalformed  ::= OBJECT IDENTIFIER{id-DocumentDefect 1}
```

The Chip Security Object might be incorrectly encoded. The Card Security Object SHOULD be used instead.

This type of defect is indicated by the following object identifier. It does not provide any parameter:

```
id-ChipSecurityMalformed  ::= OBJECT IDENTIFIER{id-DocumentDefect 2}
```

A.2.4.2 Chip Security Object Malformed

The Chip Security Object might be incorrectly encoded. The Card Security Object SHOULD be used instead.

This type of defect is indicated by the following object identifier. It does not provide any parameter:

```
id-ChipSecurityMalformed  ::= OBJECT IDENTIFIER{id-DocumentDefect 2}
```

A.2.4.3 Powerdown Required

The chip denies multiple successive authentication using the General Authentication Procedure. Either the reader MUST powerdown the chip or the document MUST be removed from the reader in between two authentications.

This type of defect is indicated by the following object identifier. It does not provide any parameter:

```
id-PowerDownReq  ::= OBJECT IDENTIFIER{id-DocumentDefect 3}
```

B Block Lists

Restricted Identification (cf. [TR-03110-2], Section 3.6) is a mechanisms that allows to generate sector-specific identifiers on the chip of the document. Those identifiers are used for the revocation of the document.

Block Lists are signed lists containing the sector-specific identifiers of revoked documents. As those identifiers are sector specific, a separate Block List is produced for every sector.

B.1 Block List Format

A Block List can be either a complete list containing all items or a delta list containing added or removed items. The block list SHOULD be provided as DER encoded `SignedData` according to [TR-03129-1] Appendix C with content type `BlockList` identified by `id-BlockList`:

```
id-BlockList OBJECT IDENTIFIER ::= {
  bsi-de applications(3) eID(2) 2}

BlockList ::= SEQUENCE{
  version      INTEGER{v2(1)},
  type         INTEGER{complete(0),added(1),removed(2)},
  listID       ListID,
  deltaBase    ListID OPTIONAL, -- required for delta lists
  finalEntries INTEGER OPTIONAL, – required for delta lists
  content      SEQUENCE OF BlockListDetails,
}

BlockListDetails ::= SEQUENCE{
  sectorID           OCTET STRING,
  sectorSpecificIDs  SEQUENCE OF OCTET STRING
}

ListID ::= OCTET STRING
```

The data type `ListID` SHALL uniquely identify the block list. The exact format of this parameter (e.g. a sequence number or a time stamp over the corresponding block list) SHALL be determined by the block list provider.

For complete block lists `deltaBase` MUST be omitted. For delta lists `deltaBase` SHALL indicate the block list used to generate the added or removed items.

The field `finalEntries` contains the number of active revocations that should be in the system after the list has been processed. If the receiving system determines, that the number of entries in its own database does not match the number of entries reported by the superior system, then the data set should be reconsolidated.

B.1.1 Block List Content

The content of `BlockListDetails` differs depending on the provider of the block list:

- If the block list is generated by the MBS it SHALL have the following content:
  - `sectorID` is the Revocation Sector.
  - `sectorSpecificIDs` are the transformed Restricted Identification public keys $PK_{ID}^{Revocation}$.

- After the DV has transformed the received block list for a specific Terminal Sector it SHALL have the following content:
  - `sectorID` is the respective Terminal Sector.
  - `sectorSpecificIDs` are the sector-specific identifiers I_{ID}^{Sector}.

The sector identifier `SectorID` SHALL be used to reference the public keys $PK^{Revocation}$ and PK_{Sector} using the service `GetSectorPublicKey` provided by the MBS and the DVs, respectively. The format of the sector identifier is defined by the Block List Provider, it is RECOMMENDED to use a hash of the public key.

All public keys in `SectorSpecificIDs` SHALL be contained as plain public key values, i.e. excluding the domain parameters. For Elliptic Curve Public Keys the uncompressed encoding MUST be used.

C Signed Data Profile

Block Lists and Defect Lists are provided as Signed Data according to [RFC 5652]. The following profile SHALL be used.

Value		Comments
SignedData		
version	m	Value = v3
digestAlgorithms	m	
encapContentInfo	m	
eContentType	m	id-DefectList or id-BlockList for a Defect List or Block List, respectively.
eContent	m	The encapsulated content.
certificates	m	The list of certificates SHALL include the certificate of the signer and SHOULD include the CSCA certificate.
crls	x	CRLs MUST be distributed separately.
signerInfos	m	It is RECOMMENDED that only one signerInfo is provided within this field.
SignerInfo	m	
version	m	The value of this field is dictated by the sid field. See [RFC 5652], Section 5.3 for rules regarding this field
sid	m	
subjectKeyIdentifier	r	It is RECOMMENDED to support this field over issuerAndSerialNumber.
digestAlgorithm	m	The algorithm identifier of the algorithm used to produce the hash value over eContent and SignedAttrs.
signedAttrs	m	DER encoded collection of attributes that MUST include attributes below.
ContentType	m	MUST match eContentType
MessageDigest	m	hash value over the encapsulated content (eContent)
SigningTime	m	Signing time
ListContentDescription	m	cf. Appendix C.1
signatureAlgorithm	m	The algorithm identifier of the algorithm used to produce the signature value, and any associated parameters.
signature	m	The signature value over signedAttrs.
unsignedAttrs	o	Other unprotected information. It is RECOMMENDED to ignore this field.

Table 3: Signed Data Profile

C.1 List Content Description

The List Content Description attribute may contain a description of the exact content of a Defect List (e.g. Internal, External, Commercial) or of a Block List (e.g. Master or Sector). The attribute SHALL be identified by id-ListContentDescription with parameter Description.

```
id-ListContentDescription{
  bsi-de applications(3) MRTD(1) 6}
}
```

```
Description ::= UTF8String
```

D WSDL and XML Scheme specifications

The corresponding XML scheme definitions and WSDL descriptions are part of this specification and provided as separate files.

Reference Documentation

CP-eID	BSI: Certificate Policy für die Country Verifying Certification Authority, eID-Anwendung, 2021
ICAO 9303	ICAO: Doc 9303: Machine Readable Travel Documents, 8th Edition, 2021
RFC 2119	Bradner, Scott: Key words for use in RFCs to indicate requirement levels, 1997
RFC 5280	Cooper, et al.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2008
RFC 5652	Housley, Russel: Cryptographic Message Syntax (CMS), 2009
RFC 5652	Housley, Russell: Cryptographic Message Syntax (CMS), 2009
TR-03110-1	BSI: TR-03110-1: Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS token, Part 1: eMRTDs with BAC/PACEv2 and EACv1, Version 2.20, 2015
TR-03110-2	BSI: TR-03110-2: Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS token, Part 2: Protocols for electronic Identification, Authentication and Trust Services (eIDAS), Version 2.21, 2016
TR-03110-3	BSI: TR-03110-3: Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS token, Part 3: Common Specifications, Version 2.21, 2016
TR-03110-3	BSI: TR-03110-3: Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS token, Part 3: Common Specifications, Version 2.21, 2016
TR-03110-4	BSI: TR-03110-4: Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS token, Part 4: Applications and Document Profiles, Version 2.21, 2016
TR-03129-1	BSI: TR-03129-1: Protocols for the Management of Certificates and CRLs in Public-Key-Infrastructures (PKIs), Part 1: Common Specifications, Version 1.4, 2022
TR-03129-2	BSI: TR-03129-2: PKIs for Machine Readable Travel Documents Part 2: Supplemental specifications for public and official authorities, 2017
TR-03129-Annex	BSI: Annex to BSI TR-03129: CMS profile for content signature, 2022