

First edition
2010-07-15

AMENDMENT 1
2019-02

**Automatic vehicle and equipment
identification — Electronic
registration identification (ERI) for
vehicles —**

**Part 4:
Secure communications using
asymmetrical techniques**

AMENDMENT 1

*Identification automatique des véhicules et des équipements —
Identification d'enregistrement électronique (ERI) pour les
véhicules —*

*Partie 4: Communications sécurisées à l'aide de techniques
asymétriques*

AMENDEMENT 1



Reference number
ISO 24534-4:2010/Amd.1:2019(E)

© ISO 2019



COPYRIGHT PROTECTED DOCUMENT

© ISO 2019

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 204, *Intelligent transport system*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

STANDARDSISO.COM : Click to view the full PDF of ISO 24534-4:2010/Amd 1:2019

Automatic vehicle and equipment identification — Electronic registration identification (ERI) for vehicles —

Part 4: Secure communications using asymmetrical techniques

AMENDMENT 1

Page 42, 6.2.10.6

Replace definition of CommissionErtErrors by

```
CommissionErtErrors ErrorCode ::= {illegalArgument | illegalVehicleId |
illegalCertificate | illegalSignature | illegalEntity | illegalDate
| notCustomized | resourceLimitExceeded | noEnciphermentCapability |
secretKeyEncryptionAlgorithmNotSupported |
publicKeyEncryptionAlgorithmNotSupported | noSigningCapability |
hashingAlgorithmNotSupported | signingAlgorithmNotSupported | otherError, .. }
```

Page 42, 6.2.11.1

Replace

```
withdrawCommissioningCode INTEGER ::= 9
```

by

```
withdrawCommissioningCode INTEGER ::= 14
```

Page 43, 6.2.11.3

Replace

```
WithdrawCommissioningResultData ::= [APPLICATION withdrawCommissioningCode ]
SEQUENCE {
withdrawn WithdrawCommissioningArgument,
historicComData HistoricComData
}
```

by

```
WithdrawCommissioningResultData ::= SEQUENCE {
withdrawn WithdrawCommissioningArgument,
historicComData HistoricComData
}
```

Page 43, 6.2.11.4

Replace definition of WithdrawCommissioningError by

```
WithdrawCommissioningErrors ErrorCode ::= {illegalArgument | illegalVehicleId  
| illegalCertificate | illegalSignature | illegalEntity | illegalDate |  
notCustomized | notCommissioned | otherError, ... }
```

Page 57, 6.2.18.3.5

Replace definition of EntityRole by

```
EntityRole ::= INTEGER {  
    topLevelCertificationAuthority      (0),  
    intermediateCertificationAuthority  (1),  
    manufacturer                        (2),  
    registrationAuthority               (3),  
    authority                           (4),  
    serviceProvider                    (5),  
    eriHolder                           (6)  
} (0..7)
```

Page 66, Annex A

Replace the annex by the following one.

Annex A (normative)

ASN.1 module

A.1 Overview

The following ASN.1 modules are specified in this normative annex:

- The normative module **ElectronicRegistrationIdentificationTransactionsModule** {iso(1) standard(0) iso24534 (24534) transactions (2) version1 (1)} in A.2.
- The informative module **AviEriDSRCData** {iso(1) standard(0) iso24534 (24534) transactions (2) example1(1) version1 (1)} in A.3.

In case the ASN.1 specifications given in this Annex are not compliant with illustrations or specifications provided elsewhere in this International standard, the specifications given in this Annex shall prevail.

The ASN.1 modules contained in this Annex will be published on <http://standards.iso.org/iso/24534/-4>.

A.2 Module ElectronicRegistrationIdentificationTransactionsModule

```
ElectronicRegistrationIdentificationTransactionsModule {iso(1) standard(0) iso24534 (24534)
transactions (2) version1 (1)}
DEFINITIONS AUTOMATIC TAGS ::= BEGIN
```

```
IMPORTS
```

```
RegistrationAuthority, VehicleId, EriData, AdditionalEriData, EntityId FROM
ElectronicRegistrationIdentificationVehicleDataModule {iso(1) standard(0) iso24534 (24534)
vehicleData (1) version1 (1)} -- ISO 24534-3
```

```
EmbeddedERIType FROM AVIAEIDSRInterfaceModule {iso (1) standard(0) iso17264(17264)
version1 (1)}
;
```

```
EriPdu ::= SEQUENCE{
    fill          BIT STRING (SIZE(6)),
    pdu           CHOICE {
        requestPdu    EriRequestPdu,
        responsePdu    EriResponsePdu
    }
}
```

```
EriRequestPdu ::= SEQUENCE {
    transactCode TRANSACTION.&transactionCode ( {EriTransactions}),
    argument TRANSACTION.&ArgumentType ( {EriTransactions} {@.transactCode}) OPTIONAL
}
```

```
EriResponsePdu ::= CHOICE {
    resultPdu          EriResultPdu,
    errorPdu           EriErrorPdu
}
```

```
EriResultPdu ::= SEQUENCE {
    transactCode TRANSACTION.&transactionCode ( {EriTransactions}),
    result TRANSACTION.&ResultType ( {EriTransactions} {@.transactCode})
}
```

```
-- warning: Constraint checking for field 'error' would not be done at runtime as
constraints applied to FixedTypeValueSetFields or VariableTypeValueSetFields are not
currently supported.
```

```

EriErrorPdu ::= SEQUENCE {
    transactCode TRANSACTION.&transactionCode ( {EriTransactions} ),
    error TRANSACTION.&ErrorCodes ( {EriTransactions} {&.transactCode} )
}

-- TRANSACTIONS
TRANSACTION ::= CLASS {
    &ArgumentType ,
    &ResultType ,
    &ErrorCodes ErrorCode OPTIONAL,
    &transactionCode TransactionCode UNIQUE
}
WITH SYNTAX {
    ARGUMENT &ArgumentType
    RESULT &ResultType
    [ERRORS &ErrorCodes]
    CODE &transactionCode
}

TransactionCode ::= INTEGER {
    tc-getEriData (1),
    tc-getAuthenticatedEriData (2),
    tc-setEriData (3),
    tc-getCiphertextHistoricEriData (4),
    tc-getCleartextHistoricEriData (5),
    tc-getPublicCertificateVerificationKeyId (6),
    tc-getPublicEnciphermentKeyErt (7),
    tc-commissionErt (8),
    tc-getCiphertextHistoricComData (9),
    tc-getCleartextHistoricComData (10),
    tc-updateAccessControlList (11),
    tc-getCiphertextAccessControlListEntry (12),
    tc-getCleartextAccessControlListEntry (13),
    tc-withdrawCommissioning (14),
    tc-getErtCapabilities (15)
} (0..255)

EriTransactions TRANSACTION ::= {getEriData | getAuthenticatedEriData | setEriData |
getCiphertextHistoricEriData | getCleartextHistoricEriData |
getPublicCertificateVerificationKeyId | getPublicEnciphermentKeyErt | commissionErt |
withdrawCommissioning | getCiphertextHistoricComData |
getCleartextHistoricComData | updateAccessControlList |
getCiphertextAccessControlListEntry | getCleartextAccessControlListEntry |
getErtCapabilities, ... }

-- Get ERI data
getEriData TRANSACTION ::= {
    ARGUMENT GetEriDataArgument
    RESULT GetEriDataResult
    ERRORS {notCustomized}
    CODE tc-getEriData
}

GetEriDataArgument ::= SEQUENCE {
    onBehalfOf EntityId OPTIONAL,
    challenge Challenge,
    includeAdditionalData BOOLEAN
}

GetEriDataResult ::= SEQUENCE {
    registrationAuthority RegistrationAuthority OPTIONAL,
    eriResultData SECURED {CleartextEriData}
}

-- Authenticate ERI data
getAuthenticatedEriData TRANSACTION ::= {
    ARGUMENT GetAuthenticatedEriDataArgument
    RESULT GetAuthenticatedEriDataResult
    ERRORS {notCustomized}
    CODE tc-getAuthenticatedEriData
}

```

```

    }

    GetAuthenticatedEriDataArgument ::= SEQUENCE {
        ertHolderCredentials      ErtHolderCredentials,
        challenge                  Challenge,
        includeAdditionalData      BOOLEAN
    }

    GetAuthenticatedEriDataResult ::= SEQUENCE {
        registrationAuthority      EntityId OPTIONAL,
        authenticateResultData     CLEAR-SECURED {CleartextEriData}
    }

    -- ERI data and ERT security flags
    CleartextEriData ::= SEQUENCE {
        eriDataOrId                EriDataOrId,
        ertSecurityStatus           ErtSecurityFlags OPTIONAL
    }

    EriDataOrId ::= CHOICE {
        vehicleId                  VehicleId,
        unsignedDatedEriData        DATED { EmbeddedERType{EriData} },
        datedAndSignedEriData       SIGNED { DATED { EmbeddedERType{EriData} },
        PrivateSignatureKey} -- BOE signed
    }

    /*
    EriData ::= SEQUENCE {
        id                          VehicleId,
        additionalEriData           OCTET STRING (CONTAINING AdditionalEriData) OPTIONAL
    }
    */

    ErtSecurityFlags ::= BIT STRING {
        flagsHaveBeenResetted      (0),
        notCommissioned             (1),
        lowSupplyVoltageIndication (2),
        highSupplyVoltageIndication (3),
        lowClockFrequencyIndication (4),
        highClockFrequencyIndication (5),
        lowTemperatureIndication    (6),
        highTemperatureIndication   (7)
    } (SIZE (0..16)) -- bit 8 .. 15 reserved for future use

    -- Set ERI data
    setEriData TRANSACTION ::= {
        ARGUMENT SetEriDataArgument
        RESULT NULL
        ERRORS {SetEriDataErrors}
        CODE tc-setEriData
    }

    SetEriDataArgument ::= CHOICE {
        clearTextArgument          ClearTextSetEriDataArgument,
        encryptedArgument           ENCRYPTED {ClearTextSetEriDataArgument}
    }

    ClearTextSetEriDataArgument ::= CHOICE {
        authenticatedEriData        BOE-AUTHENTICATED{DATED{EmbeddedERType{EriData} }},
        notAuthenticatedEriData     DATED { EmbeddedERType{EriData} }
    }

    SetEriDataErrors ErrorCode ::= {illegalArgument | illegalVehicleId | illegalCertificate |
    illegalSignature | illegalDate | notCommissioned | resourceLimitExceeded | otherError, ...
    }

    -- Retrieve historic ERI data
    getCiphertextHistoricEriData TRANSACTION ::= {
        ARGUMENT GetCiphertextHistoricEriDataArgument
        RESULT SECURED {HistoricEriData}
        ERRORS {notCustomized}
    }

```

```

CODE tc-getCiphertextHistoricEriData
}

GetCiphertextHistoricEriDataArgument ::= SEQUENCE {
    onBehalfOf          EntityId OPTIONAL,
    challenge            Challenge,
    number              INTEGER (1..int4)
}

getCleartextHistoricEriData TRANSACTION ::= {
    ARGUMENT GetCleartextHistoricEriDataArgument
    RESULT CLEAR-SECURED {HistoricEriData}
    ERRORS {notCustomized}
    CODE tc-getCleartextHistoricEriData
}

GetCleartextHistoricEriDataArgument ::= SEQUENCE {
    credentials          ErtHolderCredentials,
    challenge            Challenge,
    number              INTEGER (1..int4)
}

HistoricEriData ::= SEQUENCE {
    number              INTEGER (1..int4),
    outOf              INTEGER (1..int4),
    historicRecord      ClearTextSetEriDataArgument
}

-- Get public verification key
getPublicCertificateVerificationKeyId TRANSACTION ::= {
    ARGUMENT NULL
    RESULT KeyId
    CODE tc-getPublicCertificateVerificationKeyId
}

getPublicEnciphermentKeyErt TRANSACTION ::= {
    ARGUMENT BOE-AUTHENTICATED {VehicleId}
    RESULT PublicEnciphermentKey
    ERRORS {GetPublicEnciphermentKeyErrors}
    CODE tc-getPublicEnciphermentKeyErt
}

GetPublicEnciphermentKeyErrors ErrorCode ::= {illegalArgument | illegalVehicleId |
illegalCertificate | illegalSignature | illegalEntity | noDeciphermentCapability |
otherError, ... }

-- Commissioning
commissionErt TRANSACTION ::= {
    ARGUMENT CommissionErtArgument
    RESULT NULL
    ERRORS {CommissionErtErrors}
    CODE tc-commissionErt
}

CommissionErtArgument ::= CHOICE {
    authenticatedData    BOE-AUTHENTICATED {DATED {CommissioningData}},
    notAuthenticatedData DATED {CommissioningData}
}

CommissioningData ::= SEQUENCE {
    vehicleId            VehicleId,
    registrationAuthority EntityId,
    resetSecurityFlags    BOOLEAN,
    enciphermentKeyId     KeyId OPTIONAL,
    publicEnciphermentKeyAuthority PublicEnciphermentKey OPTIONAL,
    publicVerificationKeyCertificate ErtCertificate OPTIONAL,
    privateData ENCRYPTED {PrivateCommissioningData} OPTIONAL
}

PrivateCommissioningData ::= SEQUENCE {
    privateSignatureKeyErt PrivateSignatureKey OPTIONAL,

```

```

pin
}

CommissionErtErrors ErrorCode ::= {illegalArgument | illegalVehicleId | illegalCertificate
| illegalSignature | illegalEntity | illegalDate | notCustomized | resourceLimitExceeded |
noEnciphermentCapability | secretKeyEncryptionAlgorithmNotSupported |
publicKeyEncryptionAlgorithmNotSupported | noSigningCapability |
hashingAlgorithmNotSupported | signingAlgorithmNotSupported | otherError, ... }

-- Withdraw Commissioning
withdrawCommissioning TRANSACTION ::= {
  -- this transaction also removes all public encipherment keys
  ARGUMENT WithdrawCommissioningArgument
  RESULT SECURED {WithdrawCommissioningResultData}
  ERRORS {WithdrawCommissioningErrors}
  CODE tc-withdrawCommissioning
}

WithdrawCommissioningArgument ::= CHOICE {
  authenticatedData      BOE-AUTHENTICATED {VehicleId},
  notAuthenticatedData   VehicleId
}

WithdrawCommissioningResultData ::= SEQUENCE {
  withdrawn              WithdrawCommissioningArgument,
  historicComData        HistoricComData
}

WithdrawCommissioningErrors ErrorCode ::= {illegalArgument | illegalVehicleId |
illegalCertificate | illegalSignature | illegalEntity | illegalDate | notCustomized |
notCommissioned | otherError, ... }

-- Get historic commissioning data
getCiphertextHistoricComData TRANSACTION ::= {
  ARGUMENT GetCiphertextHistoricComDataArgument
  RESULT SECURED {HistoricComData}
  ERRORS {notCommissioned}
  CODE tc-getCiphertextHistoricComData
}

GetCiphertextHistoricComDataArgument ::= SEQUENCE {
  onBehalfOf             EntityId OPTIONAL,
  challenge              Challenge,
  number                 INTEGER (1..int4)
}

getCleartextHistoricComData TRANSACTION ::= {
  ARGUMENT GetCleartextHistoricComDataArgument
  RESULT CLEAR-SECURED {HistoricComData}
  ERRORS {notCommissioned}
  CODE tc-getCleartextHistoricComData
}

GetCleartextHistoricComDataArgument ::= SEQUENCE {
  credentials            ErthHolderCredentials,
  challenge              Challenge,
  number                 INTEGER (1..int4)
}

HistoricComData ::= SEQUENCE {
  number                 INTEGER (1..int4),
  outOf                  INTEGER (1..int4),
  historicRecord         CommissionErtArgument
}

-- Update access control list
updateAccessControlList TRANSACTION ::= {
  ARGUMENT UpdateAccessControlListArgument
  RESULT NULL
  ERRORS {UpdateAccessControlListErrors}
  CODE tc-updateAccessControlList
}

```

```

    }

UpdateAccessControlListArgument ::= CHOICE {
    authorityUpdate    SIGNED {AccessControlListUpdate, PrivateSignatureKey},
    holderUpdate       HolderAccessControlListUpdate
}

HolderAccessControlListUpdate ::= SEQUENCE {
    credentials        ErtHolderCredentials,
    entry              AccessControlListUpdate
}

AccessControlListUpdate ::= SEQUENCE {
    mode               ENUMERATED {deleteAllAndAdd (0), add (1), delete (2)} DEFAULT add,
    entry              AccessControlEntry OPTIONAL
}

AccessControlEntry ::= SEQUENCE {
    id                 EntityId,
    name               Text OPTIONAL,
    enciphermentKeyId  KeyId OPTIONAL,
    publicEnciphermentKeyReader  PublicEnciphermentKey OPTIONAL
}

UpdateAccessControlListErrors ErrorCode ::= {illegalArgument | illegalVehicleId |
illegalSignature | illegalHolderAccess | illegalDate | noEntry | resourceLimitExceeded |
noEnciphermentCapability | otherError, ... }

-- Retrieve the access control list
getCiphertextAccessControlListEntry TRANSACTION ::= {
    ARGUMENT GetCiphertextAccessControlListEntryArgument
    RESULT SECURED {AuthorityAccessControlListEntry}
    CODE tc-getCiphertextAccessControlListEntry
}

GetCiphertextAccessControlListEntryArgument ::= SEQUENCE {
    challenge          Challenge,
    number             INTEGER (1..int4)
}

getCleartextAccessControlListEntry TRANSACTION ::= {
    ARGUMENT GetCleartextAccessControlListEntryArgument
    RESULT CLEAR-SECURED {AccessControlListEntry}
    ERRORS {GetCleartextAccessControlListEntryErrors}
    CODE tc-getCleartextAccessControlListEntry
}

GetCleartextAccessControlListEntryArgument ::= SEQUENCE {
    credentials        ErtHolderCredentials,
    challenge          Challenge,
    number             INTEGER (1..int4)
}

AuthorityAccessControlListEntry ::= AccessControlListEntry (WITH COMPONENTS {...,
holderEntry ABSENT} )

AccessControlListEntry ::= SEQUENCE {
    number             INTEGER (0..int4),
    outOf              INTEGER (0..int4),
    authorityEntry      AccessControlEntry OPTIONAL,
    holderEntry        AccessControlEntry OPTIONAL
}

GetCleartextAccessControlListEntryErrors ErrorCode ::= {illegalArgument | illegalVehicleId
| illegalHolderAccess | otherError, ...}

-- Get ERT capabilities
getErtCapabilities TRANSACTION ::= {
    ARGUMENT NULL
    RESULT ErtCapabilities
    CODE tc-getErtCapabilities
}

```

```

    }

ErtCapabilities ::= SEQUENCE {
    supportedTransactions      SEQUENCE OF INTEGER,
    hashingAlgorithms          SEQUENCE OF HashingAlgorithm OPTIONAL,
    signingAlgorithms           SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
    signatureVerificationAlgorithms SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
    secretKeyEncryptionAlgorithms SEQUENCE OF SecretKeyAlgorithm OPTIONAL,
    publicKeyEncryptionAlgorithms SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
    publicKeyDecryptionAlgorithms SEQUENCE OF PublicKeyAlgorithm OPTIONAL,
    maxBitsPublicKeys           INTEGER (0..int4),
    maxOctetsPin                 INTEGER (0..int4),
    maxOctetsSetArgument         INTEGER (0..int4),
    maxNumberSetArguments        INTEGER (1..int4),
    maxNumberComArguments        INTEGER (1..int4),
    maxSizeAccessControlList     INTEGER (0..int4),
    maxNumberAuthorities          INTEGER (0..int4),
    maxNumberAddServProviders    INTEGER (0..int4),
    ertSecurityIndicationSupport ErtSecurityFlags,
    maxInteger                   INTEGER (1..int4),
    maxStringSize                INTEGER (1.. int4),
    ...
}

-- ERT tagging and authentication
SECURED {ToBeSecured} ::= CHOICE {
    authenticatedAndEncrypted ENCRYPTED {ERT-AUTHENTICATED {TAGGED {ToBeSecured}}},
    authenticated              ERT-AUTHENTICATED {TAGGED {ToBeSecured}},
    encrypted                  ENCRYPTED {TAGGED {ToBeSecured}},
    cleartext                  TAGGED {ToBeSecured}
}

CLEAR-SECURED {ToBeSecured} ::= SECURED {ToBeSecured} (WITH COMPONENTS
{authenticatedAndEncrypted ABSENT, encrypted ABSENT} )

TAGGED {ToBeTagged} ::= SEQUENCE {
    ertNumber      ErtNumber,
    challenge       Challenge OPTIONAL,
    sequenceNumber INTEGER (1..int4) OPTIONAL,
    toBeTagged      ToBeTagged
}

ERT-AUTHENTICATED {ToBeErtAuthenticated} ::= SEQUENCE {
    ertSigned          SIGNED {ToBeErtAuthenticated, PrivateSignatureKey},
    publicVerificationKeyCertificate ErtCertificate
}

ErtCertificate ::= SIGNED {ErtCertificationData, PrivateSignatureKey}

ErtCertificationData ::= SEQUENCE {
    vehicleId      VehicleId,
    publicVerificationKey PublicVerificationKey,
    signatoryId     EntityId,
    date            DATE
}

-- BOE Dates and Authentication
DATED {ToBeDated} ::= SEQUENCE {
    date          DATE,
    validThru     DATE OPTIONAL,
    issuer        EntityId,
    toBeDated     ToBeDated
}

BOE-AUTHENTICATED {ToBeBoeAuthenticated} ::= SEQUENCE {
    signedParameter SIGNED {ToBeBoeAuthenticated, PrivateSignatureKey},
    certificates     SEQUENCE SIZE(1..2) OF BoeCertificate
}

BoeCertificate ::= SIGNED {BoeCertificationData, PrivateSignatureKey}

```

```

BoeCertificationData ::= SEQUENCE {
    entityId                EntityId,
    entityRole              EntityRole,
    entityName              Text OPTIONAL,
    publicKey               Key,
    signatoryId             EntityId,
    signatoryName           Text OPTIONAL,
    signatoryRole           EntityRole,
    date                   DATE,
    validThru              DATE
}

EntityRole ::= INTEGER {
    topLevelCertificationAuthority (0),
    intermediateCertificationAuthority (1),
    manufacturer (2),
    registrationAuthority (3),
    authority (4),
    serviceProvider (5),
    eriHolder (6)
} (0..7)

-- Signing
SIGNED {ToBeSigned, PrivateSignatureKey} ::= SEQUENCE {
    toBeSigned              ToBeSigned,
    hashingAlgorithm        HashingAlgorithm DEFAULT sha1,
    signatureAlgorithm      PublicKeyAlgorithm DEFAULT ellipticCurve,
    signature               SIGNATURE {ToBeSigned, HashingAlgorithm, PublicKeyAlgorithm,
PrivateSignatureKey}
}

SIGNATURE {ToBeSigned, HashingAlgorithm, SignatureAlgorithm, PrivateSignatureKey}
::= BIT STRING (CONSTRAINED BY {
    HashingAlgorithm, -- and --
    SignatureAlgorithm, -- with --
    PrivateSignatureKey, -- applied to --
    ToBeSigned} )

Challenge ::= INTEGER (1..int4)

-- Encryption
ENCRYPTED {ToBeErtEncrypted} ::= SEQUENCE {
    enciphermentKeyId      KeyId,
    publicKeyEncryptionAlgorithm PublicKeyAlgorithm DEFAULT ellipticCurve,
    secretKeyEncryptionAlgorithm SecretKeyAlgorithm DEFAULT aes,
    encryptedKey KEY-ENCRYPTION {SecretTransactionKey, PublicKeyAlgorithm,
PublicEnciphermentKey},
    ciphertext CIPHER-TEXT {ToBeErtEncrypted, SecretKeyAlgorithm, SecretTransactionKey}
}

KEY-ENCRYPTION {KeyToBeEncrypted, PublicKeyAlgorithm, PublicEnciphermentKey} ::= BIT
STRING ( CONSTRAINED BY {
    PublicKeyAlgorithm, -- with --
    PublicEnciphermentKey, -- applied to --
    KeyToBeEncrypted -- or random bit string with same length if no public key is available
--} )

CIPHER-TEXT {ToBeEncrypted, SecretKeyAlgorithm, SecretKey} ::= BIT STRING ( CONSTRAINED BY
{
    SecretKeyAlgorithm, -- with --
    SecretKey, -- applied to --
    ToBeEncrypted -- or random bit string with same length if no public key is available
--} )

-- Encryption algorithms
HashingAlgorithm ::= ENUMERATED {
    sha1,
    ...}

PublicKeyAlgorithm ::= ENUMERATED {
    ellipticCurve,

```