INTERNATIONAL STANDARD ISO/IEC 9594-8:2014
TECHNICAL CORRIGENDUM 2

Published 2016-10-15

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

Information technology — Open Systems Interconnection — The Directory —

Part 8: Public-key and attribute certificate frameworks

TECHNICAL CORRIGENDUM 2

Technologies de l'information — Interconnexion de systèmes ouverts (OSI) — L'annuaire —

Partie 8: Cadre général des certificats de clé publique et d'attribut

RECTIFICATIF TECHNIQUE 2

Technical Corrigendum 2 to ISO/IEC 9594-8:2014 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 6, *Telecommunications and information exchange between systems.* The identical text is published as Rec. ITU-T X.509 (2012)/ Cor.2 (04/2016).

ICS 35.100.70 **Ref. No. ISO/IEC 9594-8:2014/Cor.2:2016(E)**

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks

Technical Corrigendum 2

(Covering resolution to defect reports 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419 and 420)

1) Correction of the defects reported in defect report 406

In clause 8.5.3.1 and Annex A, update the ASN.1 for ***CRLReason*** *as follows:*

```
CRLReason ::= ENUMERATED {
  unspecified          (0),
  keyCompromise        (1),
  cACompromise         (2),
  affiliationChanged   (3),
  superseded           (4),
  cessationOfOperation (5),
  certificateHold      (6),
  removeFromCRL        (8),
  privilegeWithdrawn   (9),
  aACompromise         (10),
  ... ,
  weakAlgorithmOrKey   (11) }
```

Add a new bullet point:

– **weekAlgorithm** indicates that the certificate was revoked due to a weak cryptographic algorithm and/or key (e.g., due to short key length or unsafe key generation).

In clause 8.6.2.1 and Annex A, update the ASN.1 for ***ReasonFlags*** *as follows:*

```
ReasonFlags ::= BIT STRING {
  unused               (0),
  keyCompromise        (1),
  cACompromise         (2),
  affiliationChanged   (3),
  superseded           (4),
  cessationOfOperation (5),
  certificateHold      (6),
  privilegeWithdrawn   (7),
  aACompromise         (8),
  weakAlgorithmOrKey   (9) }(SIZE(0..9,...,10))
```

2) Correction of the defects reported in defect report 407

Add the following abbreviations to clause 4:

DSA	Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
RSA	Rivest-Shamir-Adelman

Delete the old DSA abbreviation

Update 6.1 as shown

6.1 Digital signatures

Replace this clause with:

This subclause describes digital signatures in general. Sections 2 and 3 of this Directory Specification discuss the use of digital signatures within PKI and PMI specifically. This subclause is not intended to specify a specification for digital signatures in general, but to specify the means by which instances of the PKI and PMI specific data types are signed.

There are different types of digital signatures, such as RSA digital signatures, DSA digital signatures and ECDSA digital signatures.

NOTE 1 – It is not within the scope of this Specification to describe these different techniques in details, but Annex F gives a short introduction with references to more detailed specifications.

Figure 1 illustrates how a signer of instances of PKI/PMI data types (public-key certificates, attribute certificates, revocation lists, etc.) creates a digital signature and then adds that to the data before transmission. It also illustrates how the recipient of the signed data (the validator) validates the digital signature.

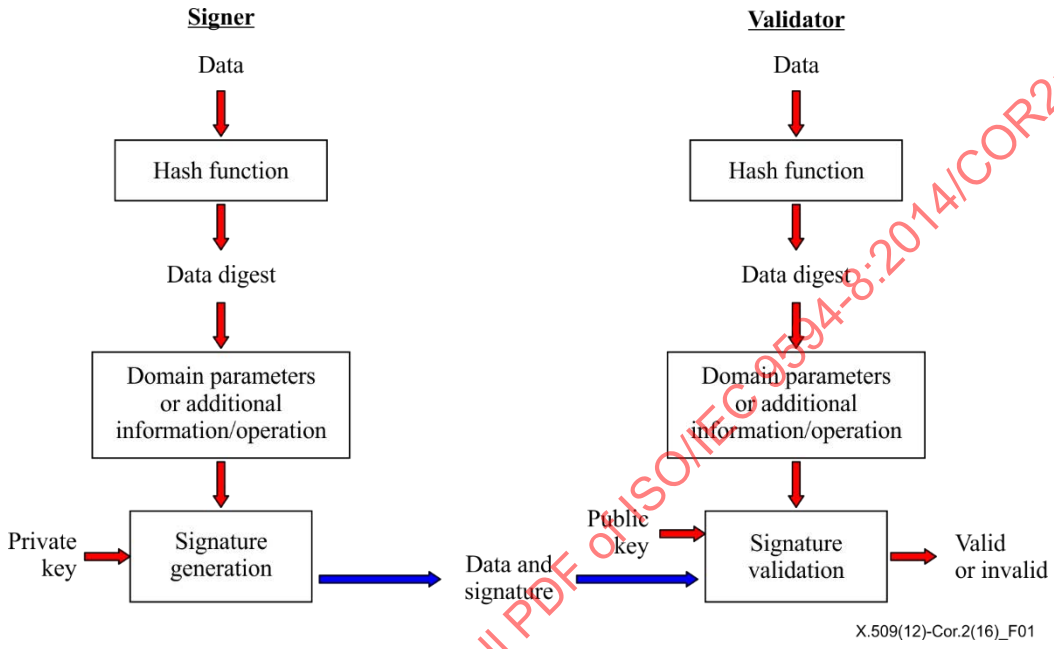

Figure 1 – Digital signature generation and validation

The digital signature signer goes through the following procedure:

1) The signer creates a hash digest over the PKI/PMI data using a secure hashing algorithm (see Annex F).
2) The hash digest is then supplemented with additional information in preparation for generating of the digital signature for improved security and for padding the hash digest to a length required by the asymmetric cryptographic function. For the RSA algorithms that can be adding some information to the hash digest and in some cases, to perform yet another hashing operation. For the DSA and ECDSA signature algorithms additional domain parameters are added.
3) The result from item 2) together with the private key of the signer and the use of a specific algorithm result in a bit string that together with an identity of the used algorithm constitute the digital signature.
4) The signature is appended to data to be signed.

Having received the data, the recipient (validator) goes through a similar procedure:

1) The validator goes through the same procedure as in steps 1) and 2) above, and if the received data is unmodified, the result will be the same as for the signer. If not, the next step will fail.
2) The result from item 1) together with the public key of the signer, the bit string of the signature and the use of a corresponding algorithm, the digital signature is evaluated as either valid or invalid.

If the digital signature proves valid, the validator has ensured that the data has not been modified and that the signer is in the position of the private key that corresponds to the public key used by the validator, i.e., the digital signature provides insurance of data integrity and authentication of the signer.

If the digital signature proves invalid, either the data has been modified or the signing private key does not corresponds to the public key used by the validator.

NOTE 2 – Data to be stored in a database or a directory may also be appended a digital signature, which then can evaluated at the retrieval of the data.

Replace clause 6.2 with:

6.2 Public-key cryptography and cryptographic algorithms

6.2.1 Formal specification of public-key cryptography

The digital signature of a data item may expressed by the following ASN.1 data type, where the **signature** component is a bit string resulting from using the appropriate signature algorithm.

```
SIGNATURE ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}},
  signature            BIT STRING,
  ... }
```

In the case where a signature is appended to the data, the following ASN.1 may be used to define the data type resulting from applying a signature to the given data type.

```
SIGNED{ToBeSigned} ::= SEQUENCE {
  toBeSigned    ToBeSigned,
  COMPONENTS OF SIGNATURE,
  ... }
```

The following data type are is not used anymore by this Specification. It may be useful in other areas and is retained for possible import by referencing specifications.

```
HASH{ToBeHashed} ::= SEQUENCE {
  algorithmIdentifier  AlgorithmIdentifier{{SupportedAlgorithms}},
  hashValue            BIT STRING (CONSTRAINED BY {
   -- shall be the result of applying a hashing procedure to the DER-encoded
   -- octets of a value of -- ToBeHashed } ),
  ... }
```

The following data types are deprecated and are not used anymore by this Specification. They are retained for possible import by referencing specifications.

```
ENCRYPTED{ToBeEnciphered} ::= BIT STRING (CONSTRAINED BY {
  -- shall be the result of applying an encipherment procedure
  -- to the BER-encoded octets of a value of -- ToBeEnciphered } )

ENCRYPTED-HASH{ToBeSigned} ::= BIT STRING (CONSTRAINED BY {
  -- shall be the result of applying a hashing procedure to the DER-encoded (see 6.2)
  -- octets of a value of -- ToBeSigned -- and then applying an encipherment procedure
  -- to those octets -- } )
```

6.2.2 Formal definitions of cryptographic algorithms

The following ASN.1 information object class is used for specifying cryptographic algorithms.

```
ALGORITHM ::= CLASS {
  &Type          OPTIONAL,
  &id            OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
  [PARMS         &Type]
  IDENTIFIED BY &id }
```

The following general data type specifies the syntax of an algorithm specification:

```
AlgorithmIdentifier{ALGORITHM:SupportedAlgorithms} ::= SEQUENCE {
  algorithm   ALGORITHM.&id({SupportedAlgorithms}),
  parameters  ALGORITHM.&Type({SupportedAlgorithms}{@algorithm}) OPTIONAL,
  ... }
```

The **algorithm** component shall be an object identifier that uniquely identifies the cryptographic algorithm being defined.

The **parameters** component, when present, shall specify the parameters associated with the algorithm. Some, but not all algorithms require associated parameters.

```
/* The definitions of the following information object set is deferred to referencing
specifications having a requirement for specific information object sets.*/
```

```
SupportedAlgorithms ALGORITHM ::= {...}
```

The elliptic curve algorithms use the **parameters** component to specify the object identifier identifying a particular curve. The following object gives a general specification for an ECC public key algorithm:

```
ecPublicKey ALGORITHM ::= {
  PARMS       SupportedCurves
  IDENTIFIED  BY id-ecPublicKey }
```

The **ecPublicKey** algorithm is defined in IETF RFC 5480 and provided here for easy reference. The associated object identifier is defined as:

```
id-ecPublicKey OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) ansi-X9-62(10045)
                                      keyType(2) 1 }

/* The definitions of the following information value set is deferred to referencing
specifications having a requirement for specific value sets.*/

SupportedCurves OBJECT IDENTIFIER ::= {dummyCurv, ...}

dummyCurv OBJECT IDENTIFIER ::= {2 5 5}
```

NOTE – The ASN.1 requires that a value set shall hold at least one value. Not to make a preference for a specific curve, a dummy value is used here that might be replaced by a referencing specifications or implementers' agreement. The shown object identifier is not otherwise used by this specification.

In clause 7.2, remove the ***AlgorithmIdentifier*** *datatype, the* ***SupportedAlgorithms*** *object set and the* ***ALGORITHM*** *object class.*

In Annex A, replace:

```
ALGORITHM ::= CLASS {
  &Type          OPTIONAL,
  &id            OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
                 [&Type]
  IDENTIFIED BY &id }
```

with:

```
ALGORITHM ::= CLASS {
  &Type          OPTIONAL,
  &id            OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
  [PARMS        &Type]
  IDENTIFIED BY &id }
```

After this new definition of ALGORITHM, add:

```
ecPublicKey ALGORITHM ::= {
  PARMS       SupportedCurves
  IDENTIFIED  BY id-ecPublicKey }

id-ecPublicKey OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) ansi-X9-62(10045)
                                      keyType(2) 1 }

/* The definitions of the following information value set is deferred to referencing
specifications having a requirement for specific value sets.*/

SupportedCurves OBJECT IDENTIFIER ::= {dummyCurv, ...}

dummyCurv OBJECT IDENTIFIER ::= {2 5 5}
```

3) Correction of the defects reported in defect report 408

Update 3.5.26 as shown:

3.5.26 end entity: Either a public-key certificate subject that uses its private key for purposes other than signing public-key certificates, or an attribute certificate holder that cannot delegate privileges of the attribute certificate, ~~that~~ but uses its attributes only to gain access to a resource.

Update 3.5.27 as shown:

3.5.27 end-entity attribute certificate: An attribute certificate issued to an entity, which then acts as an end entity within a privilege management infrastructure.

Delete 3.5.29.

Update 3.5.30 as shown:

3.5.30 end-entity public-key certificate: A public-key certificate issued to an entity, which then acts as an end entity within a public-key infrastructure.

4) Correction of the defects reported in defect report 409

Replace 3.5.2 with:

3.5.2 attribute authority (AA): An authority which assigns privilege attributes to entities by either issuing attribute certificates or including them in public-key certificates. In the latter case, the authority is also a certification authority.

5) Correction of the defects reported in defect report 410

Add the following new definition to clause 3.5 in alphabetical order, and reorder subsequent clauses accordingly:

3.5.x privilege holder: An entity that has been assigned privilege. A privilege holder may assert its privilege for a particular purpose

6) Correction of the defects reported in defect report 411

Change 11.3.5 to say:

The **`distributionPoint`** component, when present, matches if the stored attribute value contains an issuing distribution point extension and the value of this component in the presented value equals the corresponding value, in at least one name form, in that extension

7) Correction of the defects reported in defect report 412

Delete item g) of clause 8.4.1.

8) Correction of the defects reported in defect report 413

Replace 3.5.36 with:

3.5.36 indirect CRL (iCRL): A revocation list whose scope includes public-key certificates issued by one or more CAs other than the issuer of the revocation list. The same indirect CRL is also authoritative for the public-key certificates, if any, issued by the CRL issuer.

Add the following new definition to clause 3.5 in alphabetical order, and reorder subsequent clauses accordingly:

3.5.x indirect ACRL (iACRL): A revocation list whose scope includes attribute certificates issued by one or more AAs other than the issuer of the revocation list. The same indirect ACRL is also authoritative for the attribute certificates, if any, issued by the ACRL issuer.

Add new clauses 7.11 and 7.12 and renumber current clause 7.11 to clause 7.13

7.11 Uniqueness of names

A PKI is requires that CAs uniquely and unambiguously named. If CRL issuing authorities are not uniquely named, it may result in incorrect use of revocation information.

It is outside the scope of this Specification specify procedures that ensure unique and unambiguous names for CA CRL issuing authorities.

7.12 Indirect CRLs

7.12.1 Introduction

The only mechanism defined for CRL delegation by this Specification (and IETF RFC 5280) is for the public-key certificate issuing CA to include a **cRLDistributionPoint** extension in a public-key certificate and include the **cRLIssuer** component in this extension. The public-key certificate issuing CA will have do this for each certificate whose revocation status the CA wishes to delegate via CRL to a CRL issuing authority.

There is no mechanism (i.e., public-key certificate or CRL extension) for a public-key certificate issuing CA to delegate CRL issuance for all its public-key certificate to another authority using a mechanism (similar to the delegated OCSP Responder public-key certificate as specified in IETF RFC 6960).

For example, if a certificate issuing CA has issued a large number of public-key certificates and its wishes to delegate CRL issuance for all of these public-key certificate to a CRL issuing authority, the CA shall assert the **cRLIssuer** component in the **cRLDistributionPoint** extension of each of the issued public-key certificates. If the CA wishes to delegate issuance only for some of the issued public-key certificates, the CA shall assert the **cRLIssuer** component in the **cRLDistributionPoint** extension of the delegated public-key certificates and shall not assert the **cRLIssuer** component in the remaining public-key certificates that contain the **cRLDistributionPoint** extension.

The relationship of CRL delegation may be as follows:

a) A CA can delegate issuance of CRL for a given public-key certificate to multiple CRL issuance authorities. The CA might delegate to multiple authorities for the sake of redundancy by asserting multiple CRL issuers in a single distribution point in the **cRLDistributionPoint** extension or by asserting multiple distribution points in the **cRLDistributionPoint** extension with each distribution point containing one or more CRL issuer(s). Another example is a CA delegating CRL issuance to different authorities for different reason codes. In this case, the CRL Distribution Point extension must contain two or more distribution points with each distribution point containing applicable reason code(s) and CRL Issuer(s).

b) A CA can delegate issuance of CRL for different batch of public-key certificates to different CRL issuance authorities. The CA could create these batches stochastically or using a deterministic algorithm such as based on type of public-key certificate, reason code, issuance time, expiration time, subject organization, etc.

c) A CRL issuance authority can be authoritative for revocation information for public-key certificates issued by multiple CAs.

7.12.2 Indirect CRL contents

If a CRL issuance authority is a CA the CRLs it issues are authoritative for the public-key certificates issued by the CRL issuance authority as the CA and the public-key certificates whose revocation status is delegated to the CRL issuance authority. Thus, a CRL issued by a CRL issuance authority which has been delegated CRL issuance by m CAs, is authoritative for m or m + 1 CAs depending on whether the CRL issuance authority is a CA or not.

The CRL issued by the CRL issuance authority can be partitioned like any other CRLs using **distributionPoint** component of the **cRLDistributionPoint** extension. Furthermore, the CRL issuance authority may or may not choose to partition the CRL based on the public-key certificate issuer. If it chooses the former, it creates a partitioned CRL for each CA. But, the partitioned CRL discussion is outside the scope of this Specification.

Since the iCRL is authoritative for the CA(s) other that the CRL issuer, serial number alone in the CRL entry does not uniquely identify a public-key certificate that has been revoked. You also need to identify the CA that issued the public-key certificate placed on the iCRL. This is achieved by adding the **certificateIssuer** CRL entry extension. This extension shall always be flagged as critical to ensure that the relying parties process it and associated the CRL entry with the appropriate public-key certificate.

If each entry on a CRL contained the **certificateIssuer** extension (which is a directory distinguished name), it would make the CRL size large. Thus, in order to reduce the CRL size, the iCRL issuing authority should sort the CRL entries by issuing CA. Using this approach, only the first public-key certificate appearing on the CRL for a given CA needs to contain the **certificateIssuer** extension. All subsequent entries are assumed to for the same public-key certificate issuing CA until another **certificateIssuer** CRL entry extension is encountered. To further reduce the size of the CRL, if the iCRL issuing authority is a CA, it should contain its revoked public-key certificate first, obviating the need for **certificateIssuer** extension for any of its certificates.

NOTE 1 – The following example illustrates the use of iCRLs. In the example, there is single CRL issuing authority that issues revocation lists for multiple CAs. The issuing distribution point extension is present in that CRL is and flagged as critical. The **indirectCRL** component of this extension is set to **TRUE**. If the CRL issuing authority name is same as the name for one the CAs it serves, entries should then be placed first on the CRL without the certificate issuer extension. Entries for other CAs are kept together and the first CRL entry for a particular CA has included the certificate issuer extension flagged as critical.