

Representing Digital Signatures Using the CMS (Cryptographic Message Syntax) Format

Version 1.0

1 Annotation

This document defines the method of using the CMS format to represent data elements that form a digital signature.

2 References

RFC 2560 Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C., X.509 Internet Public Key Infrastructure: Online Certificate Status Protocol - OCSP. June 1999.

RFC 2630 Housley, R., Cryptographic Message Syntax. June 1999.

DAKÜP Digitaalalkirja kontrolli üldpõhimõtted (“General Principles of Digital Signature Verification”)

APA Ajatepliteenuse protokollid ja andmevormingud (“The Protocols and Data Formats of the Timestamping Service”)

3 Terms and Definitions

4 Introduction

CMS (*Cryptographic Message Syntax*) [RFC2630] defines the `SignedData` format that enables to structurally represent signed data, the signature, and miscellaneous user-defined attributes.

This document defines the methods of representing a digital signature according to requirements of the specification [DAKÜP], using the CMS format.

To store with the signature the data that CMS does not deal with (for example, timestamps, validity confirmations of freshness tokens), the signed and unsigned attributes of the `SignerInfo` data structure, which represents the signature, are used. According to the CMS specification, the data to be signed are combined with signed attributes inside the `signAttrs` field. The resulting data collection will be signed. Data elements that need to be protected with the signature must be saved as signed attributes. Data elements that need not to be protected with the signature must be saved as unsigned attributes.

5 Data Formats

5.1 Message Digest of Signing Certificate

To exactly identify the signing certificate, the signed attribute `id-certificateDigest` is added to the signature. The attribute has the following identifier:

```
id-certificateDigest OBJECT IDENTIFIER ::= { ... }
```

The attribute's single value contains the `CertificateDigest` ASN.1 type.

```
CertificateDigest ::= MessageImprint
```

The `MessageImprint` data structure is defined in the [APA] specification and represents the message digest with the hash function's identifier used to create it.

The given structure is produced after hashing the signing certificate's DER encoding (with type and length bytes).

This attribute is obligatory.

5.2 Freshness Token

If timestamps are used for determining the signing time, then a freshness token must be added to the signature.

The freshness token is inserted into unsigned attribute `id-freshnessToken`. The attribute's identifier is:

```
id-freshnessToken OBJECT IDENTIFIER ::= { ... }
```

The attribute's single value is the `LinkedTimestampToken` data structure (defined in [APA]) representing the freshness token.

Additionally, the message digest of the token's DER encoded `TimestampInfo` field is inserted into the `id-freshnessDigest` attribute. The attribute's identifier is:

```
id-freshnessDigest OBJECT IDENTIFIER ::= { ... }
```

The attribute's single value is the `MessageImprint` structure that contains the message digest of the freshness token's hashed `TimeStamPInfo` field.

Both attributes are mandatory if timestamps are used to determine the signing time; otherwise, the attributes are unused.

5.3 Signature's Timestamp

If timestamps are used to determine the signing time, then the signature's timestamp must be added to the signature.

A signature's timestamp is added to the DER encoding of the `TimestampInput` data structure.

```
TimestampInput ::= SEQUENCE {  
    signatureDigest MessageImprint,
```

```
    crlDigest          [0] MessageImprint OPTIONAL,  
    ocspDigest         [1] MessageImprint OPTIONAL  
}
```

The `TimestampInput` data structure contains the following fields:

`signatureDigest` – The message digest of a byte stream (without the OCTET STRING type and length bytes) that is the value of the `signature` field inside `SignerInfo`.

`crlDigest` – A message digest of the revocation list, in DER encoding. The list confirms the validity of the signing certificate and it can be stored in the `SignedData` structure's `crls` field.

`ocspDigest` – A message digest of the `BasicOCSPResponse` structure, in DER encoding. The structure is inside the OCSF response and confirms the validity of the signing certificate. The OCSF response can be stored inside the unsigned attribute `id-ocspResponse` (see chapter 5.4).

The timestamped `TimestampInput` data structure will be saved into the unsigned attribute `id-timestampInput`. The attribute's identifier is:

```
id-timestampInput OBJECT IDENTIFIER ::= { ... }
```

The attribute's single value is the `TimestampInput` data structure, the timestamping input.

The signature's timestamp is inserted into the unsigned `id-signatureTimestamp` attribute. It has the following identifier:

```
id-signatureTimestamp OBJECT IDENTIFIER ::= { ... }
```

The attribute's single value is the `LinkedTimestampToken` data structure (defined in [APA]) representing the signature's timestamp.

Both attributes are mandatory if timestamps are used to determine the signing time, otherwise, the attributes are unused.

5.4 OCSP Response

The `BasicOCSPResponse` structure inside the OCSF response that confirms the validity of the signing certificate, can be stored in the unsigned `id-ocspResponse` attribute. It has the following identifier:

```
id-ocspResponse OBJECT IDENTIFIER ::= { ... }
```

The attribute's single value is the `BasicOCSPResponse` data structure (defined in [RFC2560]) representing the contents of an OCSF response.