International Standard

ISO/IEC 23264-2

# Information security — Redaction of authentic data —

## Part 2: Redactable signature schemes based on asymmetric mechanisms

*Sécurité de l'information — Rédaction de données authentifiées —*

*Partie 2: Schémas de signature éditable basés sur des mécanismes asymétriques*

First edition
2024-08

Reference number
ISO/IEC 23264-2:2024(en)

## Contents

## Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

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 document 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 or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *Information security, cybersecurity and privacy protection*.

A list of all parts in the ISO/IEC 23264 series can be found on the ISO and IEC websites.

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 and www.iec.ch/national-committees.

# Introduction

This document specifies cryptographic mechanisms to redact authentic data where the redactable attestation scheme is based on asymmetric mechanisms.

Attestation schemes, in particular digital signature schemes or message authentication codes, can be used to provide data integrity and data origin authentication. Redactable attestation can be used to blank out parts, herein called fields, of an attested message without invalidating the attestation on the remaining contents of the message. This redaction process requires a redaction key. The redaction key computationally does not reveal the attestation key, by which schemes can allow for public redactions. Any other modification of the document (e.g. redaction of other message parts, or insertion/modification of any parts) will invalidate the attestation. Schemes can have specific additional security properties, which are described in ISO/IEC 23264-1. The achievable properties for each scheme are stated in this document.

Redactable attestation schemes are a basic building block in many privacy-preserving applications, such as privacy-preserving data sharing or authentication, where a party may decide to forward only necessary information to a receiver, while the latter is still assured that the received information was previously attested, for example, by a public authority.

The objective of the ISO/IEC 23264 series is to remedy existing incompatibilities or inconsistently defined properties found in academic literature, and to ease the real-world adoption of this technology. Specifically, the goal of this document is to focus on algorithms that enable the authenticity-preserving redaction of general data structures like sets or ordered lists based on asymmetric cryptography. It adheres to the common terminology and description of cryptographic properties for redactable attestation schemes given in ISO/IEC 23264-1.

The ISO/IEC 23264 series complements ISO/IEC 27038, which specifies the redaction of digital documents without considering the authenticity of the data.

This document contains the following algorithms based on asymmetric cryptography:

— generic construction from signature schemes and hash-functions

— scheme SBZ02-MERSAProd

— scheme BBDFFKMOPPS10

— scheme DPSS15

— scheme MHI06

— scheme MIMSYTI05

# Information security — Redaction of authentic data —

# Part 2:
# Redactable signature schemes based on asymmetric mechanisms

## 1 Scope

This document specifies cryptographic mechanisms to redact authentic data. The mechanisms described in this document offer different combinations of the security properties defined and described in ISO/IEC 23264-1. For all mechanisms, this document describes the processes for key generation, generating the redactable attestation, carrying out redactions and verifying redactable attestations.

This document contains mechanisms that are based on asymmetric cryptography using three related transformations:

— a public transformation defined by a verification key (verification process for verifying a redactable attestation),

— a private transformation defined by a private attestation key (redactable attestation process for generating a redactable attestation), and

— a third transformation defined by the redaction key (redaction process) allowing to redact authentic information within the constraints set forth during generation of the attestation such that redacted information cannot be reconstructed.

This document contains mechanisms which, after a successful redaction, allow the attestation to remain verifiable using the verification transformation and attest that non-redacted fields of the attested message are unmodified. This document further details that the three transformations have the property whereby it is computationally infeasible to derive the private attestation transformation, given the redaction and or the verification transformation and key(s).

## 2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 23264-1, *Information security — Redaction of authentic data — Part 1: General*

## 3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 23264-1 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

**3.1**
**balanced tree**
height-balanced tree
*tree* (3.13) in which the heights of the immediate subtrees of each node differ at most by one

[SOURCE: ISO/IEC 2382:2015, 2121638, modified — notes to entry have been removed.]

**3.2**
**binary tree**
ordered *tree* (3.13) in which each node has at most two other nodes that are directly subordinate

[SOURCE: ISO/IEC 2382:2015, 2121636, modified — notes to entry have been removed.]

**3.3**
**balanced binary tree**
*binary tree* (3.2) which is a *balanced tree* (3.1)

**3.4**
**collision-resistant hash-function**
*hash-function* (3.6) satisfying the following property: it is computationally infeasible to find any two distinct inputs which map to the same output

Note 1 to entry: Computational feasibility depends on the specific security requirements and environment (see ISO/IEC 10118-1:2016, Annex C).

[SOURCE: ISO/IEC 10118-1:2016, 3.1, modified — reference to ISO/IEC 10118-1:2016, Annex C has been added in the note.]

**3.5**
**hash-code**
string of bits which is the output of a *hash-function* (3.6)

Note 1 to entry: The literature on this subject contains a variety of terms that have the same or similar meaning as hash-code. Modification Detection Code, Manipulation Detection Code, digest, hash-result, hash-value and imprint are some examples.

[SOURCE: ISO/IEC 10118-1:2016, 3.3]

**3.6**
**hash-function**
function which maps strings of bits of variable (but usually upper bounded) length to fixed-length strings of bits, satisfying the following two properties:

— for a given output, it is computationally infeasible to find an input which maps to this output;

— for a given input, it is computationally infeasible to find a second input which maps to the same output

Note 1 to entry: Computational feasibility depends on the specific security requirements and environment. See ISO/IEC 10118-1:2016, Annex C.

[SOURCE: ISO/IEC 10118-1:2016, 3.4, modified — reference to ISO/IEC 10118-1:2016, Annex C has been added in the note.]

**3.7**
**height**
maximum number of nodes in any path leading from the *root node* (3.10) to a *leaf node* (3.8)

[SOURCE: ISO/IEC 2382:2015, 2121637, modified — “leaf node” has replaced "terminal node" in the definition; notes to entry have been removed.]

**3.8**
**leaf node**
node that has no *subordinate node* (3.11)

[SOURCE: ISO/IEC 2382:2015, 2121489, modified — term "leaf node" has replaced the original terms "leaf" and "terminal node".]

**3.9**
**parent node**
node to which at least one other node is directly subordinate

[SOURCE: ISO/IEC 2382:2015, 2121488, modified — notes to entry have been removed.]

**3.10**
**root of a tree**
**root node**
node of a *tree* (3.13) that has only *subordinate nodes* (3.11)

**3.11**
**subordinate node**
node at the other end of an outgoing arc; a node may have zero, one, or more subordinates

[SOURCE: ISO/IEC 9804:1998, 3.6.64]

**3.12**
**subtree**
part of a *tree* (3.13) including a node and all its *subordinate nodes* (3.11)

[SOURCE: ISO/IEC 2382:2015, 2121634, modified —notes to entry have been removed.]

**3.13**
**tree**
data structure containing nodes that are linked together hierarchically by oriented arcs with at most one parent node for each node, and with only one *root node* (3.10)

[SOURCE: ISO/IEC 2382:2015, 2121633, modified – “by oriented arcs” has been added to the definition; notes to entry have been removed.]

## 4 Symbols and conventions

### 4.1 Symbols

Throughout this document, the following symbols are used.

| | |
|---|---|
| $adm$ | description of redacted or original admissible changes |
| $adm'$ | description of redacted admissible changes |
| $ak$ | attestation key |
| $att$ | redactable or redacted attestation |
| $att'$ | redacted attestation |
| $m$ | redacted or original message |
| $m'$ | redacted message |
| $m_1, \dots, m_n$ | individual field |

| | |
|---|---|
| *mod* | description of modification instructions |
| *n* | number of fields |
| *pk* | public verification key used internally by asymmetric signature algorithm or accumulator scheme |
| *rk* | redaction key |
| *root* | value assigned as content to the node forming the root of a tree |
| *sk* | secret signing key used internally by asymmetric signature algorithm or accumulator scheme |
| *tag* | string used to mark messages or message fields |
| *vk* | verification key |
| *Z* | set of one or more domain parameters |
| *Σ* | output of a digital signature scheme as defined in ISO/IEC 14888-1 |
| *reject*, *accept* | output of a digital signature verification |

## 4.2 Conventions

A triple of message, attestation, and admissible changes is denoted as (*m, att, adm*). In the same way a triple of redacted message, a redacted attestation and redacted admissible changes is denoted by $(m', att', adm')$.

A specific value of a symbol *sym* is denoted as *sym**, or *sym*** in order to differentiate their specific values. An equality is stated using the equals sign =.

EXAMPLE 1 The statement that *m*=m*** means that the contents of *m** are equal to the contents of *m***.

The symbol || denotes a concatenation of strings.

When the symbol || is applied to values which are not represented as strings, the values are required to be converted into a string before they are concatenated. See for example Reference [25] for a conversion of an integer into a string.

The symbol $\lfloor a \rfloor$ indicates the largest integer not exceeding *a*, following the definition of the floor function specified in ISO/IEC 15444-1.

EXAMPLE 2 $\lfloor -2{,}46 \rfloor = -3$, $\left\lfloor \frac{1}{3} \right\rfloor = 0$

The symbol $a / b$ denotes the result of the division of *a* by *b*.

This document uses set notation and symbols to denote set-like operations:

| | |
|---|---|
| \ | operation *set minus*, i.e. $A \setminus B$ denotes the contents of set *A* without contents of set *B* |
| ∪ | operation *set union*, i.e. $A \cup B$ denotes all contents from set *A* together with those of set *B* |
| ⊆ | statement *sub set or equal*, i.e. $A \subseteq B$ denotes that all contents from set *A* are contained as contents in set *B* |
| $\|A\|$ | denotes the size of the set *A*, i.e. the number of elements contained in *A* |
| ∅ | empty set |

Where applicable, the output of a hash-function is interpreted as an integer.

## 5 General

This document adheres to the common terminology and description of cryptographic properties for redactable attestation schemes given in ISO/IEC 23264-1.

Redactable attestation schemes provide data integrity and data origin authentication. While redactable attestation can be used to allow a redactor to blank out parts, herein called fields, of an attested message without invalidating the attestation on the remaining contents of the message, any other modification of the document (e.g. redaction of other message parts, or insertion/modification of any parts) will invalidate the attestation. This redaction process requires a redaction key. The redaction key computationally does not reveal the attestation key, by which schemes can allow for redactions that are either public or by a party other than the attestor. The verification key is cryptographically linked to the attestation key and allows verification of the origin of the attested data.

The schemes can have specific additional security properties, which are described in ISO/IEC 23264-1. The achievable properties for each scheme are stated in this document.

This document describes algorithms that enable the authenticity-preserving redaction of general data structures like sets or ordered lists based on asymmetric cryptography.

This document contains the following algorithms based on asymmetric cryptography:

— Generic construction from signature schemes and hash-functions

— Scheme SBZ02-MERSAProd

— Scheme BBDFFKMOPPS10

— Scheme DPSS15

— Scheme MHI06

— Scheme MIMSYTI05

Finally, this document contains the following annexes:

— Annex A, which provides object identifiers which shall be used to identify the mechanisms defined in this document.

— Annex B, which contains an overview of the different properties as defined and described in ISO/IEC 23264-1 that are achieved by the different algorithms contained in this document.

— Annex C, which lists criteria for inclusion in this document.

— Annex D, which gives numerical examples to understand and check implementations of the algorithms contained in this document.

## 6 Generic construction from signature schemes and hash-functions

### 6.1 Parameters

The generic construction makes use of the following parameters:

— digital signature scheme as defined in ISO/IEC 14888-1;

— collision-resistant hash-function Hash as defined in ISO/IEC 10118-1;

— security parameter $\lambda$.

The security parameter $\lambda$ shall be met by all involved algorithms, especially for the hash-function and digital signature scheme.

## 6.2 Construction

### 6.2.1 Key generation process

The key generation algorithm of the redactable attestation scheme consists of the following two procedures:

a) generate the set of domain parameters *Z*;

b) generate the attestation key *ak*, verification key *vk* and redaction key *rk* as follows:

   1) use the key generation process of the digital signature scheme, such that the digital signature scheme's signature key serves as the attestation key *ak*,

   2) use the digital signature scheme's verification key as the verification key *vk*,

   3) use the digital signature scheme's verification key also as the redaction key *rk*, such that *rk* = *vk.*

The security parameter $\lambda$ is taken into account such that the digital signature scheme offers at least the indicated security strength.

### 6.2.2 Redactable attestation process

This process takes the following inputs:

— set of domain parameters $Z$

— attestation key $ak$

— message $m$ consisting of $n$ fields $m_1,\ldots,m_n$

— admissible changes $adm=\{1,\ldots,n\}$

The process Split() to split the message $m$ into fields is generally out of scope of this document. However, the process Split() shall be defined in such a way that fields can be transformed into the message and vice versa, in a reproducible way every time this conversion is necessary, i.e. if Split($m$)= $m_1,\ldots,m_s$ then Split$^{-1}$( $m_1,\ldots,m_s$ ) = $m$. The scheme protects the content of each field and their order.

The process requires the generation of random data. Refer to ISO/IEC 18031 to generate this securely.

The process does not allow the specification of admissible changes, so by default all fields of the message ( $m_1,\ldots,m_n$ ) are admissible and can be redacted by anyone. In order to make this clear for the user, the input for *adm* shall contain all field indices, i.e. $adm=\{1,\ldots,n\}$ .

The process is as follows:

a) Generate a Merkle tree[17] as follows: Generate a binary tree which is also a balanced tree, henceforth referred to as balanced binary tree, of sufficient height such that it has *k* leaf nodes with 2*n*> *k* ≥ *n*.

b) Choose a uniformly random $\lambda$-bit $tag_{msg}$ for the message; and choose *n* random $\lambda$-bit tags $tag_i$, one for each field of *m*. No $tag_i$ shall contain only zeros, denoted as $tag_i \neq 0^{\lambda}$.

c) For each $i=1,\ldots,n$, compute the hash-code $h_i = \mathrm{Hash}\left(tag_{msg} \| m_i \| tag_i\right)$ using a collision-resistant hash-function.

d) Initialize the Merkle tree using $h_1,\ldots,h_n$ as values for the $n$ left-most leaf nodes of the balanced binary tree, use the empty string for all remaining $k-n$ leaf nodes.

e) Calculate the Merkle tree's root, denoted as *root*, using the collision-resistant hash-function Hash by computing the value for each parent node in the tree as Hash (*left-subordinate-node-value*||*right-subordinate-node-value*), where *left-subordinate-node-value* contains the value assigned to the left subordinate node and *right-subordinate-node-value* contains the value of the right subordinate node.

f) Use the digital signature scheme's signature process on inputs:

   1) message: $\left(root, tag_{msg}, n\right)$;

   2) set of domain parameters: *Z;*

   3) signature key mapped from the attestation key: *ak*.

   Receive as output the signature $\Sigma$.

The process outputs:

— redactable attestation $att = \left(\Sigma, n, tag_{msg}, (tag_1, \ldots, tag_n)\right)$

### 6.2.3 Redaction process

This process takes the following inputs:

— set of domain parameters $Z$;

— message *m* composed of *n* fields denoted as $m_1, \ldots, m_n$;

— redaction key $rk$;

— admissible changes $adm$;

— modification instructions $mod$ that are in accordance with the admissible changes $adm$, i.e. $mod \subseteq adm$.

The process is as follows:

a) Verify that $att = \left(\Sigma, n, tag_{msg}, (tag_1, \ldots, tag_n)\right)$ is a valid attestation on *m* under the verification key $vk = rk$ and abort if this is not the case.

b) Set $m' = m$ and $att' = att$.

c) Adjust the admissible changes to no longer contain fields to be redacted, i.e. *adm'* = *adm* \ *mod*.

d) For all $i \in mod$:

   1) Compute the hash-code $h_i = \text{Hash}\left(tag_{msg} \| m_i \| tag_i\right)$ using the collision-resistant hash-function.

   2) Replace the content of $m_i$ with $h_i$, i.e. set $m' = (m_1, \ldots, m_{i-1}, h_i, m_{i+1}, \ldots, m_n)$.

   3) Set $tag_i = 0^\lambda$ to indicate that this field has been redacted, i.e. modify the attestation to $att' = \left(\Sigma, n, tag_{msg}, \left(tag_1, \ldots, tag_{i-1}, 0^\lambda, tag_{i+1}, \ldots, tag_n\right)\right)$.

The process outputs:

— redacted message *m';*

— redacted attestation *att';*

— redacted admissible changes *adm'* = *adm* \ *mod.*

### 6.2.4 Verification process

The process takes the following inputs:

— set of domain parameters $Z$ ;

— verification key $vk$ ;

— redacted or original message $m=(m_1,\ldots,m_n)$ ;

— redacted or redactable attestation $att=\left(\Sigma,n,tag_{msg},(tag_1,\ldots,tag_n)\right)$ ;

— original or redacted admissible changes $adm$ .

The process is as follows:

a) Generate a Merkle tree[17] as follows: generate a balanced binary tree of sufficient height such that it has $k$ leaf nodes with $2n>k\geq n$.

b) For each $i=1,\ldots,n$, compute the hash-code $h_i=\text{Hash}\left(tag_{msg}\,\|\,m_i\,\|\,tag_i\right)$ if $tag_i\neq 0^\lambda$ using a collision-resistant hash-function; and if $tag_i=0^\lambda$ then set $h_i$ to the value supplied as $m_i$ as it has been redacted previously, which corresponds to the value of $\text{Hash}\left(tag_{msg}\,\|\,m_i\,\|\,tag_i\right)$.

c) Initialize the Merkle tree using $h_1,\ldots,h_n$ as values for the $n$ left-most leaf nodes of the balanced binary tree, use the empty string for all remaining $k-n$ leaf nodes.

d) Calculate the value for the root (denoted as $root$ ) of the Merkle tree.

e) Use the digital signature's verification process on inputs:

   — set of domain parameters $Z$ ;

   — verification key mapped from the verification $vk$

   — message: $\left(root,tag_{msg},\ n\right)$ ;

   — signature: $\Sigma$ .

   Receive from the digital signature's verification process an output $o\in\{accept,reject\}$ .

The process outputs:

— The final verification outcome $o$ .

NOTE 1 The redactable attestation scheme specified in 6.2.1 to 6.2.4 satisfies the following properties: unforgeability, privacy, detectability of redactions and mergeable. See Annex B for further details.

NOTE 2 Security proofs for unforgeability and privacy are found in the Reference [12]. Detectability of redactions and mergeability are shown in Reference [15]. See Annex B for further details.

NOTE 3 This scheme was originally introduced by R. Steinfeld, L. Bull, and Y. Zheng in 2001.[12]

## 7 Scheme SBZ02-MERSAProd

### 7.1 Parameters

The SBZ02-MERSAProd scheme makes use of the following parameters:

— collision-resistant hash-function Hash as defined in ISO/IEC 10118-1;

— security parameter $\lambda$.

The security parameter $\lambda$ shall be met by all involved algorithms for the hash-function and taken into account during the choice of the domain parameters and keys.

## 7.2 Construction

### 7.2.1 Key generation process

The key generation algorithm of the redactable attestation scheme consists of the following two procedures:

a) generate the set of domain parameters *Z*;

b) generate the attestation key $ak$ and verification key $vk$ and redaction key $rk$ as follows:

   1) The verification key $vk$ consists of a unique RSA (Rivest, Shamir, and Adleman) modulus $N = p \cdot q$ generated according to ISO/IEC 9796-2 and a list $e_1, \ldots, e_l$ of public exponents which are pairwise co-prime and co-prime to $(p-1)\cdot(q-1)$. Here, *l* shall be equal to or larger than the number of fields in the message being attested.

   2) The attestation key *ak* consists of the modulus *N* and the secret exponents $d_1, \ldots, d_l$ such that for all $i = 1, \ldots, l$ it holds that $d_i \equiv e_i^{-1}$ modulo $(p-1)\cdot(q-1)$.

   3) The verification key also serves as redaction key, such that $rk = vk$.

The security parameter $\lambda$ shall be taken into account when choosing above parameters, such that the indicated security strength is achieved.

NOTE In case the number of exponents is larger than the number of fields in the message, i.e. in the case of $n < l$, some of the *l* exponents are not facilitated.

### 7.2.2 Redactable attestation process

This process takes the following inputs:

— set of domain parameters $Z$;

— attestation key $ak$ consisting of the *l* secret exponents $d_1, \ldots, d_l$;

— a message $m$ consisting of *n* fields $m_1, \ldots, m_n$ in no particular order, such that each field includes a unique index number $i$ with $1 \le i \le l$;

— admissible changes *adm* represented as two sets containing the unique index number corresponding to the fields' index numbers: the first set $adm_{\text{fix}}$ containing the indices of fields that are not admissible to redaction and the second set $adm_{\text{red}}$ containing indices of fields admissible to redactions.

NOTE 1 Reference [12] does not define in detail how to efficiently encode the admissible changes.

NOTE 2 This algorithm does not directly protect the order of the fields, as the index number $i$ for each field is only required to map to a specific exponent in the verification key; the order of those exponents is not protected inside the algorithms presented in this document.

The process Split() to split the message $m$ into fields is generally out of scope of this document. However, the process Split() shall be defined in such a way that fields can be transformed into the message and vice versa, in a reproducible way everytime this conversion is necessary, i.e. if Split($m$)= $m_1, \ldots, m_s$ then Split$^{-1}$($m_1, \ldots, m_s$) = $m$. The scheme protects the contents of each field, but not their order.

The process requires the generation of randomness. Refer to ISO/IEC 18031 to generate this securely.

The process is as follows:

a) Generate a random bit string $tag_{CES}$ of length $\lambda$.

b) For each $i=1,\ldots,n$, compute a hash-code $h_i$ from the concatenation of the content of each field $m_i$ together with the *adm* and the $tag_{CES}$ using the collision-resistant hash-function: $h_i = \mathrm{Hash}(adm \| tag_{CES} \| n \| i \| m_i)$.

NOTE 3 How to represent numbers as strings for concatenation is outside the scope of this document.

c) Use the function trans to transform the output of the collision-resistant hash-function to numbers that can be used as input to the forthcoming RSA operation in step d) and e).

NOTE 4 Reference [12] does not define in detail how to encode the output of the hash-function for use with RSA.

NOTE 5 A compatible transform function is the optimal asymmetric encryption padding (OAEP) as standardized in PKCS#1 v2.2 and RFC 8017[29] and in ISO/IEC 18033-2.

d) For each $i=1,\ldots,n$, sign the $\mathrm{trans}(h_i)$ with a different exponent to obtain a signature per field: $s_i = (\mathrm{trans}(h_i))^{d_i}$ modulo $N$.

e) Compute $\Sigma$ as the product of all fields' signatures $s_i$: $\Sigma = \prod_{i=1}^{n} s_i$ modulo $N$

The process outputs:

— redactable attestation $att = (\Sigma, n, tag_{CES})$.

### 7.2.3 Redaction process

This process takes the following inputs:

— set of domain parameters $Z$;

— message $m$ composed of $n$ fields denoted as $m_1,\ldots,m_n$ such that each field includes a unique index number $i$ that corresponds to the indices to the exponents in the verification key;

— attestation *att* of the form $att = (\Sigma, n, tag_{\mathrm{CES}})$;

— redaction key $rk$ containing the list $e_1,\ldots,e_l$ of public exponents;

— admissible changes $adm$;

— modification instructions $mod$, which are represented as a set of the unique indices $i$ that corresponds to the indices of the fields that will be redacted, that are according to the admissible changes $adm$, i.e. $mod \subseteq adm_{red}$.

The process is as follows:

a) Verify that $att = (\Sigma, n, tag_{CES})$ is a valid attestation on *m* under the verification key $vk = rk$ and abort if this is not the case.

b) Check that modification instructions $mod$ are a subset or equal to $adm$.

c) Set *m′* to those fields that shall remain, i.e. remove all $m_i$ if $i \in mod$.

d) Let *X* be the set of the indices of all fields in *m′*.

e) Compute for all $k \in X$, using the Chinese-Remainder-Theorem (CRT), the coefficients $c_k$ such that:

1) $c_k \equiv 1$ modulo $e_k$ and

2) $c_k \equiv 0$ modulo $e_i$ for those $i \in \{1,\ldots,k-1,k+1,\ldots,n\}$.

f) For each $k \in X$ compute the hash-code $h_k = \mathrm{Hash}(adm \| tag_{CES} \| n \| k \| m_k)$.

g) Use the same function trans as during the attestation process [7.2.2, step c)] to transform the output of the collision-resistant hash-function to numbers that can be used as input to the forthcoming RSA operation [in step h)]; compute $h_i$=trans($h_i$) for $i = 1,2,\dots,n$.

h) For each $k \in X$ compute $s_k = \Sigma^{c_k} / \prod_{i=1,2,\dots,n} h_i^{\lfloor c_k / e_i \rfloor}$ modulo $N$.

i) Using the $s_k$ computed in h) for the remaining fields, compute the product $\Sigma'$ of all signatures $\Sigma' = \prod_{k \in X} s_k$ modulo $N$.

The process outputs:

— redacted attestation $att' = (\Sigma', n, tag_{CES})$;

— redacted message $m'$;

— admissible changes $adm$.

NOTE 1 The redaction process does not adjust the initially attested number of fields of the message, denoted as $n$.

NOTE 2 To calculate the coefficients in step e), any algorithm that solves the equations related to the CRT can be used.

NOTE 3 In step c), the indices corresponding to the remaining $m_i$ do not change and in particular they will still use the same corresponding exponent $e_i$.

NOTE 4 In step h), the division is a division modulo $N$, and not over the integers, while the division in the exponent is an integer division rounded down.

NOTE 5 The redaction process does not support consecutive redactions (i.e. redacting again an already redacted attestation) due to the current described handling of indices.

### 7.2.4 Verification process

The process takes the following inputs:

— set of domain parameters $Z$;

— verification key $vk$ containing the list $e_1, \dots, e_l$ of public exponents;

— redacted or original message $m = (m_1, \dots, m_n)$ such that each field includes a unique index number $i$ that corresponds to the indices of the exponents in the verification key $vk$;

— redacted or redactable attestation $att = (\Sigma, n, tag_{CES})$;

— admissible changes $adm$ consisting of $adm_{fix}$ and $adm_{red}$.

The process is as follows:

a) Let $X$ be the set of the indices of all fields of the message $m$.

b) Check if the indices from $X$ are corresponding to the $adm$ by:

1) checking if $adm_{fix}$ is a subset of or equal to the set $X$, and

2) checking if the set $X$ is a subset of or equal to the union of $adm_{fix}$ and $adm_{red}$.

If any check fails the verification outcome is invalid, i.e. set $o = reject$ and abort.

c) Recompute the hash-code using the concatenation of the content of each field $m_i$ together with the $adm$ and the $tag_{CES}$ using the collision-resistant hash-function: $h_i = \text{Hash}(adm \| tag_{CES} \| n \| i \| m_i)$.

d) Use the same function trans as during the attestation process to transform the output of the collision-resistant hash-function to numbers capable as input to the RSA operation [in step f)]; compute $h_i$ = trans($h_i$).

e) Compute $e = \prod_{i \in X} e_i$ .

f) Compute $r = \prod_{i \in X} h_i^{\frac{e}{e_i}}$ modulo $N$ .

g) If $r = \Sigma^e$ then set $o = accept$ otherwise set $o = reject$ .

The process outputs:

— the final verification outcome $o$ .

NOTE 1 The redactable attestation scheme specified in 7.2.1 to 7.2.4 satisfies the following properties: unforgeability, privacy, detectability of redactions, and disclosure control. See Annex B for further details.

NOTE 2 Security proofs for unforgeability and privacy are found in the original work.[12] Detectability of redactions and mergeability are shown in.[15] See Annex B for disclosure control and for further details.

NOTE 3 This scheme was originally introduced by R. Steinfeld, L. Bull, and Y. Zheng in 2001;[12] in the original work it is named MERSAProd or MERP.

## 8 Scheme BBDFFKMOPPS10

### 8.1 Parameters

The BBDFFKMOPPS10 scheme makes use of the following parameters:

— digital signature scheme as defined in ISO/IEC 14888-1;

— security parameter $\lambda$.

The security parameter λ shall be met by all involved algorithms for the signature scheme.

### 8.2 Construction

#### 8.2.1 Key generation process

The key generation algorithm of the redactable attestation scheme consists of the following two procedures:

a) generate the set of domain parameters $Z$;

b) generate the attestation key $ak$ and verification key $vk$ and redaction key $rk$ as follows:

   1) use the key generation process of the digital signature scheme, such that the digital signature scheme's signature key maps to the attestation key $ak$,

   2) the digital signature scheme's verification key maps to the verification key $vk$ ,

   3) the digital signature scheme's verification key also maps to the redaction key $rk$ , such that $rk = vk$ .

The security parameter $\lambda$ is taken into account such that the digital signature scheme offers at least the indicated security strength.

#### 8.2.2 Redactable attestation process

This process takes the following inputs:

— set of domain parameters $Z$ ;

— attestation key $ak$ consisting of the secret key of the digital signature algorithm;

— message $m$ encoded as a tree $T$ with $b$ arcs $A=\{a_1,\ldots,a_b\}$ between $t$ nodes $V=\{v_1,\ldots,v_t\}$. The tree's arcs are directed; pointing from a node denoted as parent to a node denoted as a subordinate node. All arcs' subordinate nodes (if more than one) are ordered, and in Clause 8 the ordered sequence of $q$ subordinate nodes of node $p$ is denoted as $VL_p=\{v_{p,1},\ldots,v_{p,q}\}$. The ordering relation is denoted as a is-left-of relation $R_p\subseteq VL_p\times VL_p$, which yields that $(v_{p,i},v_{p,j})\in R_p$ if and only if $i<j$. One single node shall be distinguishable from the others as it has no parent node and is denoted *root*. The ordered nodes, together with the arcs between them, form the message to become attested.

— admissible changes $adm=\{1,\ldots,t\}$.

The scheme protects the content of each field and their ordering within the tree structure, but not directly a non-tree-structured linear message. To use such a scheme for non-tree-structured message, a transform from the linear message onto a tree shall be provided. The encoding of the tree and the process Split() to split the message $m$ into arcs and nodes are both out of scope of this document, however the process Split() shall be defined in such a way that nodes, arcs and their ordering can be transformed into the message and vice versa in a reproducible way every time this conversion is necessary, i.e. if Split($m$)= $\{a_1,\ldots,a_b\},\{v_1,\ldots,v_t\},\{R_1,\ldots,R_t\}$ then $\text{Split}^{-1}(\{a_1,\ldots,a_b\},\{v_1,\ldots,v_t\},\{R,\ldots,R_t\})=m$.

The process requires the generation of random data. Refer to ISO/IEC 18031 to generate this securely.

The process does not allow admissible changes to be specified. Therefore, by default, a leaf node of the tree, i.e. a vertex that has no subordinate nodes, can always be redacted in this scheme.

In order to make this clear for the user, the input for *adm* shall contain all $t$ nodes as indices, i.e. $adm=\{1,\ldots,t\}$ ; however, redacting a non-leaf node requires previously redacting the nodes of the subtree below it.

The process is as follows:

a) For each $v\in V$, generate a random bit string $tag_v$ of length $\lambda$.

b) Initialize the attestation *att* as empty.

c) Traverse the tree $T$ starting from the node that is the root of the tree using a post-order traversal algorithm. During this post-order traversal, perform the following action for each arc $a=(v,w)\in A$:

  1) Set $m_a$ to the value of $0\|v\|tag_v\|w\|tag_w$.

  2) Use the digital signature scheme's signature process on inputs:

    i) $m_a$;

    ii) set of domain parameters $Z$;

    iii) signature key mapped from the attestation key *ak*.

  Receive as output the signature $\Sigma_a$.

  3) Concatenate the resulting signature to the front of current value of the attestation *att* to form the latest value of *att*, i.e. $att=\Sigma_a\|att$.

  NOTE 1 Step c) is performed with the intention to attest all arcs and the content of the nodes and to store the generated digital signatures in a serialized form by traversing the tree in a fixed manner, i.e. post-order traversal.

  NOTE 2 The following is an example for a recursive algorithm that performs a post-order traversal of the complete tree $T$ starting at the root by calling the following algorithm with the root node; the algorithm gets as additional input an action to perform on each node or each arc:

  d) If the node given as input has subordinate nodes, then the left-most subordinate node that has not been marked as visited yet is traversed, and the post-order function is called recursively with this node as an input.

  e) If the node given as input has no more subordinate nodes or all subordinate nodes are marked as visited, then the node is marked as visited and the results of performing the action given as input on the contents of the node, or on the last traversed arc that led to the node, are output, e.g. the contents of the nodes that are connected via this arc.

In post-order, the recursion ends with the action being applied on the root.

f) Traverse the tree $T$ starting from the node that is the root using a post-order traversal algorithm. For each node $p \in V$ that has more than one subordinate node, perform the following actions:

Let $VL_p$ contain the ordered list of the subordinate nodes of the node $p$ as the set $VL_p = \left(v_{p,1}, \ldots, v_{p,q}\right)$.

Let $RL_p$ contain all is-left-of relations for all members of the subordinate nodes of node $p$, i.e. for all members of the set $VL_p$.

For each member $\left(v_{p,i}, v_{p,j}\right)$ of the relations-set $RL_p$, perform the following steps:

1) Set $m_v$ to the value of $1 \| v_{p,i} \| tag_{v_{p,i}} \| v_{p,j} \| tag_{v_{p,j}}$.

2) Use the digital signature scheme's signature process on inputs:

   i) $m_v$;

   ii) set of domain parameters $Z$;

   iii) signature key mapped from the attestation key $ak$.

Receive as output the signature $\Sigma_v$.

3) Concatenate the resulting signature to the front of the current value of the attestation *att* to form the latest value of *att*, i.e. $att = \Sigma_v \| att$.

NOTE 3 Step d) is performed with the intention to sign the is-left-of order relation among all subordinate nodes of a single node. This is then performed repeatedly for all nodes in the tree that have further subordinate nodes to attest the tree's ordering and to store the generated digital signatures in a serialized form by traversing the tree in a fixed manner, i.e. post-order.

g) For the node $v$ that is the root of tree $T$, set $m_r$ to the value of $v \| tag_v$.

Use the digital signature scheme's signature process on inputs:

i) $m_r$;

ii) set of domain parameters $Z$;

iii) signature key mapped from the attestation key $ak$.

Receive as output the signature $\Sigma_r$.

Concatenate the resulting signature to the front of the current value of the attestation *att* to form the latest value of *att*, i.e. $att = \Sigma_r \| att$

h) Concatenate all random tags for all nodes $V$ of the tree to the back of the current value of the attestation *att* to form the latest value of *att*, i.e. $att = att \| tag_1 \| \ldots \| tag_{|V|}$.

The process outputs:

— attestation *att*.

### 8.2.3 Redaction process

This process takes the following inputs:

— set of domain parameters $Z$;

— a message $m$ encoded as a tree $T$ with $b$ directed arcs $A = \{a_1, \ldots, a_b\}$ between $t$ nodes $V = \{v_1, \ldots, v_t\}$;

— attestation *att* of the form $att = (\Sigma_r \| \Sigma_{v,1} \| \ldots \| \Sigma_{v,t} \| \Sigma_{a,1} \| \ldots \| \Sigma_{a,b} \| tag_1 \| \ldots \| tag_t)$;

— redaction key $rk$;

— modification instructions $mod$ which consists of a single node $v_{\text{mod}}$ that has no subordinate nodes, i.e. a leaf node of the tree.

The process is as follows:

a) Verify that $att = (\Sigma_r \| \Sigma_{v,1} \| \ldots \| \Sigma_{v,t} \| \Sigma_{a,1} \| \ldots \| \Sigma_{a,b} \| tag_1 \| \ldots \| tag_t)$ is a valid attestation on $m$ under the verification key $vk = rk$ and abort if this is not the case.

b) Remove from *att* the random value corresponding to the node to be removed $tag_{mod}$.

c) Remove from *att* the one value of $\Sigma_{a,x}$ which corresponds to the arc *a* where $a = (v_Z, v_{\text{mod}})$ where $v_Z$ is the parent node of $v_{\text{mod}}$.

d) Remove from *att* each value of $\Sigma_{v,y}$ which corresponds to a relation between nodes containing the node $v_{\text{mod}}$, i.e. either $(v_{\text{mod}}, v_{p,x}) \in R_p$ or $(v_{p,x}, v_{\text{mod}}) \in R_p$.

e) If the node $v_{\text{mod}}$ is the root and there are no subordinate nodes, remove $\Sigma_r$ from *att*.

The process outputs:

— redacted message *m′* encoded as a tree without the node $v_{\text{mod}}$ resulting in a tree with $V \setminus v_{\text{mod}}$ and $A \setminus (v_z, v_{\text{mod}})$;

— redacted attestation *att′*.

### 8.2.4 Verification process

This process takes the following inputs:

— set of domain parameters $Z$;

— a message $m$ encoded as a tree *T* with *b* directed arcs $A = \{a_1, \ldots, a_b\}$ between $t$ nodes $V = \{v_1, \ldots, v_t\}$

— verification key $vk$

— attestation *att* of the form $att = (\Sigma_r \| \Sigma_{v,1} \| \ldots \| \Sigma_{v,t} \| \Sigma_{a,1} \| \ldots \| \Sigma_{a,b} \| tag_1 \| \ldots \| tag_t)$ on $m$.

The process is as follows:

a) Parse *att* to retrieve the signature over the nodes $v$ that forms the root of tree $T$ as $\Sigma_r$ and the corresponding random as $tag_v$.

b) Set $m_r$ to the value of $v \| tag_v$.

c) Use the digital signature's verification process on inputs:

   1) set of domain parameters $Z$;

   2) verification key mapped from the verification $vk$;

   3) message $m_r$;

   4) signature $\Sigma_r$.

   Receive an output $o_{root} \in \{accept, reject\}$ from the digital signature's verification process.

d) If $o_{root}$ returns $accept$ then set $o = accept$, otherwise set $o = reject$ and stop.

e) Traverse the tree $T$ starting from the node that is the root using a post-order traversal algorithm. For each node $p \in V$ that has more than one subordinate node, perform the following actions:

Let $VL_p$ contain the ordered list of the subordinate nodes of the node $p$ as the set $VL_p = \{v_{p,1}, \ldots, v_{p,q}\}$.

Let $RL_p$ contain all is-left-of relations for all members of the list of subordinate nodes of node $p$, i.e. for all members of the set $VL_p$.

For each member $(v_{p,i}, v_{p,j})$ of the relations-set $RL_p$, perform the following actions:

1) Parse the attestation *att* to retrieve the signature for that relation as $\Sigma_{v,x}$ and the corresponding random values as $tag_{v_{p,i}}$ and $tag_{v_{p,j}}$;

2) Set $m_v$ to the value of $1 \| v_{p,i} \| tag_{v_{p,i}} \| v_{p,j} \| tag_{v_{p,j}}$;

3) Use the digital signature scheme's verification process on inputs:

   i) message $m_v$;

   ii) set of domain parameters $Z$;

   iii) verification key mapped from the verification key *vk*;

   iv) signature $\Sigma_{v,x}$.

Receive an output $o_{v,x} \in \{accept, reject\}$ from the digital signature's verification process.

4) If $o_{v,x}$ returns $accept$ then set $o = accept$, otherwise set $o = reject$ and stop.

NOTE 1 Step e) is performed with the intention to verify the is-left-of order relation among all subordinate nodes of a single node. This is then performed repeatedly for all nodes in the tree that have subordinate nodes, i.e. to verify the attestation of the order and the contents of all nodes.

f) Traverse the tree $T$ starting from the node that is the root using a post-order traversal algorithm. For each edge $a = (v, w) \in A$, perform the following actions:

1) Parse *att* to retrieve the signature over edge as $\Sigma_{a,y}$ and the corresponding random values as $tag_v$ and $tag_w$.

2) Set $m_a$ to the value of $0 \| v \| tag_v \| w \| tag_w$.

3) Use the digital signature scheme's signature process on inputs:

   i) $m_a$;

   ii) set of domain parameters $Z$;

   iii) signature $\Sigma_{a,y}$;

   iv) verification key mapped from the verification key *vk*.

Receive an output $o_{a,y} \in \{accept, reject\}$.

4) If $o_{a,y}$ returns $accept$ then set $o = accept$, otherwise set $o = reject$ and stop.

NOTE 2 Step f) is performed with the intention to verify the digital signatures over the arcs among all nodes and their contents to verify the attestation of the arcs and contents of all nodes in the tree.

The process outputs:

— The final verification outcome $o$.

NOTE 3 The redactable signature scheme specified in 8.2.1 to 8.2.4 satisfies the following properties: unforgeability, privacy, undetectability of redactions, and mergeability. See Annex B for further details.

NOTE 4 Security proofs for unforgeability, privacy and undetectability of redactions are found in Reference [4]. Mergeability has been shown later in Reference [15]. See Annex B for further details.

NOTE 5 This scheme was originally introduced by C. Brzuska, H. Busch, O. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder in 2010.[4]

## 9 Scheme DPSS15

### 9.1 Parameters

The DPSS15 scheme makes use of the following parameters:

— digital signature scheme as defined in ISO/IEC 14888-1;

— collision-resistant hash-function $\mathrm{Hash}$ mapping to odd numbers;

— security parameter $\lambda$.

NOTE A collision-resistant hash-function $\mathrm{Hash}$ mapping to odd numbers can be built from any collision-resistant hash-function $\mathrm{Hash}'$ as defined in ISO/IEC 10118-1 by defining $\mathrm{Hash}(x)=2\cdot\mathrm{Hash}'(x)+1$.

Some processes require the generation of random data. Refer to ISO/IEC 18031 to generate this securely.

The security parameter λ shall be met by all involved algorithms for the hash-function, digital signature scheme and the RSA accumulator.

### 9.2 Subroutine: RSA Accumulators

The redactable attestation scheme described in 9.3 uses a so-called accumulator scheme consisting of four mechanisms as a building block.

An accumulator scheme generally consists of four algorithms:

— $\mathrm{AKGen}$: to generate a keypair consisting of a secret key and a public key;

— $\mathrm{AEval}$: a secret-key-based algorithm to generate the accumulation value and auxiliary information from a set of input elements;

— $\mathrm{AWitCreate}$: a secret-key-based algorithm to produce a so-called witness for an element and an accumulation value; and

— $\mathrm{AVerify}$: a public-key-based algorithm that given a witness, an element and an accumulation value, verifies that the given element was in the set of elements accumulated into the given accumulation value using $\mathrm{AEval}$ and the secret key corresponding to the public key used for the verification.

The following describes the internal behavior of one instantiation of this mechanism, named RSA Accumulator, in detail.

— The key generation mechanism, denoted by $\mathrm{AKGen}$, takes as input the security parameter and performs the following steps:

  — Generate an RSA modulus $N=pq$ according to ISO/IEC 9796-2, such that:

    — $p$ and $q$ are distinct safe primes of sufficient length to achieve the strength indicated by the security parameter;

    — $p$ (and respectively $q$) is safe if $p=2\cdot p'+1$, where $p'$ is also prime;

  — set $\varphi = (p\text{-}1) \cdot (q\text{-}1)$.

  The mechanism returns a secret key $sk=\varphi$ and a corresponding public key $pk=N$.

— The accumulation mechanism, denoted by $\mathrm{AEval}$, takes as input a secret key $sk$, corresponding public key $pk$, and a set of elements $\{x_i\}_{i=1}^{k}$ to accumulate, and performs the following step:

  — Draw a random element $a$ in $\mathrm{Z}_N$ such that $2 \leq a < N - 1$.

  The mechanism returns an accumulation value $acc = a$ and auxiliary information $aux$ which is the empty string.

  NOTE 1 By drawing the value $a$ uniformly at random, the correct distribution required for security of the scheme is achieved independently of the elements to be accumulated (see Reference [15]). A verifiable witness can only be computed efficiently with knowledge of the secret key $sk=\varphi$.

— The witness generation mechanism, denoted by $\mathrm{AWitCreate}$, takes as inputs a secret key $sk$, corresponding public key $pk$, an accumulation value $acc$, auxiliary information $aux$, and an element $x$ which has been accumulated into $acc$, and performs the following steps:

  — Compute $x' = \mathrm{Hash}(x)$.

  — Compute $w = acc^{x'^{-1} \, modulo \, \varphi} \, modulo \, N$.

  The mechanism outputs a witness $wit = w$.

— The accumulator verification mechanism, denoted by $\mathrm{AVerify}$, takes as inputs a public key $pk$, an accumulator value $acc$, a witness $wit$, and an element $x$, and performs the following steps:

  — Compute $g = wit^{\mathrm{Hash}(x)} \, modulo \, N$.

  — If $g = acc$, set $o = accept$, and $o = reject$ otherwise.

  The mechanism outputs the single bit $o$.

NOTE 2 The above scheme was originally introduced by H. C. Pöhls and K. Samelin in 2014.[15]

NOTE 3 The redactable attestation scheme in 9.3 is agnostic of the concrete choice of the accumulator scheme as long as certain cryptographic properties are satisfied.[5] The above scheme was chosen because of its proven security properties and for efficiency reasons.

NOTE 4 The accumulator mechanism in 9.2 does not require any auxiliary information *aux*. However, the redactable attestation scheme in 9.3 is agnostic of the concrete choice of the accumulator scheme as long as certain cryptographic properties are satisfied.[5] Thus, the RSA Accumulator outputs an empty string as the auxiliary information in order to be aligned.

## 9.3 Construction

### 9.3.1 Key generation process

The key generation algorithm of the redactable attestation scheme consists of the following procedures:

a) generate the set of domain parameters $Z$;

b) generate the attestation key $ak$ and verification key $vk$ and redaction key $rk$ as follows:

  1) Using the key generation process of the digital signature scheme, compute a secret and public key pair ($sk_{DSS}$, $vk_{DSS}$).

  2) Using the key generation process of the accumulator scheme (see 9.2), compute two secret and public key pairs ($sk'_{acc}$, $pk'_{acc}$) and ($sk''_{acc}$, $pk''_{acc}$), respectively.

  3) Compute the attestation key as $ak = (sk_{DSS}, sk'_{acc}, sk''_{acc}, pk'_{acc}, pk''_{acc})$ and the verification and redaction keys as $rk = vk = (vk_{DSS}, pk'_{acc}, pk')$ $rk{=}vk{=}(vk_{DSS}, pk'_{acc}, pk''_{acc})$.

The security parameter $\lambda$ is taken into account such that the digital signature scheme and the accumulator offer at least the indicated security strength.

### 9.3.2 Redactable attestation process

This process takes the following inputs:

— set of domain parameters $Z$,

— attestation key $ak$ of the form $ak = (sk_{DSS}, sk'_{acc}, sk''_{acc}, pk'_{acc}, pk''_{acc})$,

— message $m$ consisting of $n$ fields $m_1, \ldots, m_n$, where the message fields $m_i$ are pairwise distinct,

NOTE 1 If the requirement on the distinctness cannot be guaranteed, it can easily be enforced by padding each message field with uniformly random bit string of length $2\lambda$.

— admissible changes $adm$ containing those fields that cannot be redacted.

The process Split() to split the message $m$ into fields is generally out of scope of this document. However, the process Split() shall be defined in such a way that fields can be transformed into the message and vice versa, in a reproducible way everytime this conversion is necessary, i.e. if Split($m$)= $m_1, \ldots, m_s$ then Split$^{-1}$($m_1, \ldots, m_s$) = $m$. The scheme protects the content of each field.

The process requires the generation of random data. See ISO/IEC 18031 to generate this securely.

The process is as follows:

a) For each $i = 1, \ldots, n$ perform the following steps:

   1) Draw a uniformly random bit string $r_i$ of length $2\lambda$.

   2) Compute $(acc'_i, aux') = \mathrm{AEval}\left((sk'_{acc}, pk'_{acc}), \{r_j\}_{j=1}^{i}\right)$.

   3) For each $j = 1, \ldots, i$, compute $wit'_{ij} = \mathrm{AWitCreate}\left((sk'_{acc}, pk'_{acc}), acc'_i, aux', r_j\right)$.

   4) Let $WIT'_i = (wit'_{i1}, \ldots, wit'_{ii})$.

b) Compute $(acc'', aux'') = \mathrm{AEval}\left((sk''_{acc}, pk''_{acc}), \{(m_i \| acc'_i \| r_i)\}_{i=1}^{n}\right)$.

c) For each $i = 1, \ldots, n$, compute $wit''_i = \mathrm{AWitCreate}((sk''_{acc}, pk''_{acc}), acc'', aux'', (m_i \| acc'_i \| r_i))$.

d) Set $adm' = \{(m_i \| acc'_i \| r_i)\}_{m_i \in adm}$.

e) Using the digital signature schemes' signature process on inputs:

   1) message $(acc'', adm')$;

   2) set of domain parameters $Z$;

   3) signature key $sk_{DSS}$;

   receiving as output a digital signature $\Sigma_{DSS}$.

f) Compute $att = \left(\Sigma_{DSS}, acc'', (acc'_i, WIT'_i, r_i, wit''_i)_{i=1}^{n}, adm\right)$

The process outputs the attestation $att$.

NOTE 2 The accumulator mechanism in 9.2 does not require any auxiliary information *aux*. However, the redactable attestation scheme in 9.3 is agnostic of the concrete choice of the accumulator scheme as long as certain cryptographic properties are satisfied.[5] Thus, the auxiliary information is used throughout 9.3 in order to stay agnostic for reasons of cryptographic agility.

### 9.3.3 Redaction process

This process takes the following inputs:

— set of domain parameters $Z$;

— a message $m$ of the form $m=(m_1,\dots,m_n)$;

— a redaction key $rk$ of the form $rk=(pk_{DSS},pk'_{acc},pk''_{acc})$;

— attestation $att$ of the form $att=\left(\Sigma_{DSS},acc'',(acc'_i,WIT'_i,r_i,wit''_i)_{i=1}^{n},adm\right)$ on $m$;

— original or redacted admissible changes $adm$ consisting of a list of message fields;

— modification instructions $mod$ consisting of a set of indices of message fields in $m$ to be redacted.

The process is as follows:

a) Let $L$ be a list containing all indices of $mod$ sorted in descending manner.

b) For each $i\in L$, perform the following steps:

   1) Delete the message field $m_i$ from $m$, thereby reducing the indices of all message fields with a higher index by $1$.

   2) For each $j=i+1,\dots,n$ delete $wit'_{ji}$ from $WIT'_j$, thereby reducing the indices of all entries with a higher index in the list by $1$.

   3) Delete $(acc'_i,WIT'_i,r_i,wit''_i)$ from the corresponding list in $att$, thereby reducing the indices of all entries with a higher index in this list by $1$.

   4) Adjust $n$ to the new length of the message by setting $n=n-1$.

c) Define the redacted message $m'=m$. Admissible changes are not altered, hence the redacted admissible changes are equal to $adm$, and define the redacted attestation $att'$ as the result after the above deletions from $att$ as performed in b).

The process outputs:

— redacted message $m'$;

— redacted attestation $att'$;

— redacted admissible changes $adm$.

### 9.3.4 Verification Process

This process takes the following inputs:

— set of domain parameters $Z$;

— a message $m$ of the form $m=(m_1,\dots,m_n)$;

— a verification key $vk$ of the form $vk=(pk_{DSS},pk'_{acc},pk''_{acc})$;

— attestation $att$ of the form $att=\left(\Sigma_{DSS},acc'',(acc'_i,WIT'_i,r_i,wit''_i)_{i=1}^{n},adm\right)$ on $m$.

The process is as follows:

a) Set $o=accept$.

b) Set $adm'=\{(m_i\,\|\,acc'_i\,\|\,r_i)\}_{m_i\in adm}$

c) Verify the correctness of $\Sigma_{DSS}$ by executing the digital signature scheme's verification process on inputs:

1) message $(acc'', adm')$,

2) set of domain parameters $Z$,

3) verification key $pk_{DSS}$,

4) signature $\Sigma_{DSS}$,

receiving an output $o_{DSS}$.

d) If $o_{DSS} = reject$, set $o = reject$.

e) For each $i = 1, \ldots, n$, perform the following computations:

1) Verify that there is a correct witness for $(m_i \| acc'_i \| r_i)$ by computing $o_{tmp} = \text{AVerify}(pk''_{acc}, acc'', wit''_i, (m_i \| acc'_i \| r_i))$. If $o_{tmp} = reject$, set $o = reject$.

2) For each $j = 1, \ldots, i$, verify that there is a correct witness for $r_j$ in $acc'_i$ by computing $o_{tmp} = \text{AVerify}(pk'_{acc}, acc'_i, wit'_{ij}, r_j)$, where $WIT'_i = (wit'_{i1}, \ldots, wit'_{ii})$. If $o_{tmp} = reject$, set $o = reject$.

f) Set $o = reject$, if $adm' \cap \{(m_i \| acc'_i \| r_i)\}_{i=1}^{n} \neq adm'$, as this indicates that message fields that are non-redactable according to $adm$ have been removed.

g) Verify that each $m_j \in adm$ is also contained in $m$ in the same order of occurrence, potentially interleaved with other fields, to ensure that the non-redactable message fields according to $adm$ still occur in the correct order. If this is not the case, set $o = reject$.

NOTE 1 This last step can easily be implemented using a loop.

h) Output the final verification outcome $o$.

NOTE 2 The redactable signature scheme specified in 9.3.1 to 9.3.4 satisfies the following properties: unforgeability, privacy, undetectability of redactions, and disclosure control. See Annex B for further details.

NOTE 3 The property of mergability is not shown in the original work but possible. See Annex B for further details.

NOTE 4 This scheme was originally introduced by D. Derler, H. C. Pöhls, K. Samelin, and D. Slamanig in 2015.[5]

## 10 Scheme MHI06

### 10.1 Parameters

The MHI06 scheme makes use of the following parameters:

— Groups $G_1$, $G_2$, $G_T$ of prime order $p$ equipped with a bilinear pairing $e$: $G_1 \times G_2 \rightarrow G_T$ defined via a pairing-friendly elliptic curve as specified in ISO/IEC 15946-5;

NOTE ISO/IEC 15946-5 uses additive notation for elliptic curve operations, e.g. $h = aG$, whereas this clause uses multiplicative notation following the initial work, e.g. $h = g^a$.

— Generator $g_2$ of the group $G_2$;

— Hash-function H: $\{0,1\}^* \rightarrow G_1$ mapping strings to elements of $G_1$ such as the function PHF1 specified in ISO/IEC 18033-5;

— Security parameter $\lambda$.

The security parameter $\lambda$ shall be met by all involved algorithms for the hash-function and taken into account during the choice of domain parameters and keys.

## 10.2 Construction

### 10.2.1 Key generation process

The process requires the generation of random data. Refer to ISO/IEC 18031 to generate this securely.

The key generation process consists of the following two procedures:

a) Generate the domain parameters $Z$

b) Generate the attestation key $ak$, the verification key $vk$, and the redaction key $rk$ as follows:

1) Pick a random $x$ in the range $0 < x < p$ and set the attestation key to $x : ak = x$

2) Compute the verification key as $vk = g_2^x$

3) Set the redaction key equal to the verification key: $rk = vk$

4) The security parameter $\lambda$ is taken into account such that, in the groups $G_1$, $G_2$ generated as part of the domain parameters, the computational co-Diffie-Hellman problem (given $h \in G_1$ and $g_2$, $g_2^x \in G_2$ for random $0 < x < p$, compute $h^x \in G_1$) provides at least the indicated security strength.

NOTE Redaction does not require a dedicated redaction key $rk$; the verification key $vk$ is sufficient for this purpose.

### 10.2.2 Redactable attestation process

The redactable attestation process takes the following inputs:

— domain parameters $Z$;

— attestation key $ak$;

— a message consisting of $n$ fields $m = (m_1, \ldots, m_n)$, each identified by a unique index $i$ in the range $1 \le i \le n$;

— admissible changes $adm$ consisting of a subset of the indices $1,\ldots, n$ (i.e. $adm \subseteq \{1,\ldots, n\}$) corresponding to the fields for which redaction or disclosure is admissible.

NOTE The process treats the fields $m_1, \ldots, m_n$ of the input message as an unordered set and will not enforce the order in which these fields appear in the input. In particular, the attestation generated for a message $m^*$ will be a valid attestation for any other message $m^{**}$ consisting of the same message fields, but in a different order.

The process Split() to split the message $m$ into fields is generally out of scope of this document. However, the process Split() shall be defined in such a way that fields can be transformed into the message and vice versa, in a reproducible way everytime this conversion is necessary, i.e. if Split($m$)= $m_1, \ldots, m_s$ then Split$^{-1}$($m_1, \ldots, m_s$) = $m$. The scheme protects the content of each field, but not their order.

The process requires the generation of random data. Refer to ISO/IEC 18031 to generate this securely.

The process is as follows:

a) Pick a random bit string $tag$ of length $\lambda$.

b) Compute the hash-code $h_0 \in G_1$ of the chosen $tag$ : $h_0 = \mathrm{H}(tag)$.

c) Compute a signature $\sigma_0 \in G_1$ for $tag$: $\sigma_0 = h_0^{ak}$.

d) For each $i = 1, \ldots, n$, compute the hash-code $h_i \in G_1$ of *tag* concatenated with the message field $m_i$: $h_i = \mathrm{H}(tag \,||\, m_i)$.

e) For each $i = 1, \ldots, n$, compute a signature $\sigma_i \in G_1$ for $m_i$: $\sigma_i = h_i^{ak}$.

f) Combine signatures $\sigma_0, \sigma_1, \ldots, \sigma_n$ into the joint signature $\Sigma = \prod_{j=0}^{n} \sigma_j \in G_1$.

g) Set $S$ to be the set of signatures $\sigma_i$ for which $i \in adm$: $S = \{\sigma_i \mid i \in adm\}$.

The process outputs:

— redactable attestation $att = (tag, \Sigma, S)$;

— redaction key $rk$, which is equal to the verification key $vk$.

The encoding of *adm* and *att* shall ensure that for each $i \in adm$, the corresponding message field $m_i$ and signature $\sigma_i \in S$ can be identified, as this is required by the redaction process.

The process ensures that any message field $m_i$ for which $i \notin adm$, cannot be redacted from $m$ without invalidating the attestation *att*.

### 10.2.3 Redaction process

The redaction process takes the following as inputs:

— domain parameters $Z$;

— redaction key $rk$;

— message $m$ consisting of $n$ fields $m_1, \ldots, m_n$, each identified by a unique index $i$ in the range $1 \le i \le n$;

— attestation $att = (tag, \Sigma, S)$ for the message $m$;

— admissible changes $adm$ consisting of a subset of the indices $1, \ldots, n$ (i.e. $adm \subseteq \{1, \ldots, n\}$) corresponding to the message fields for which redaction or disclosure is admissible;

— modification instructions $mod$ consisting of two disjoint sets, $mod_{\mathrm{red}} \subseteq adm$ and $mod_{\mathrm{dis}} \subseteq adm$: the set $mod_{\mathrm{red}}$ specifies the message fields to be redacted, and the set $mod_{\mathrm{dis}}$ specifies the message fields that shall always be disclosed and for which further redaction will be prohibited.

The process is as follows:

a) Verify that $att$ is a valid attestation of $m$ by using the verification process in 10.2.4 with input $vk$, $m$, and $att$. If the verification output is $o = reject$, abort.

b) Verify that:

1) for each $i \in adm$, there is a corresponding $\sigma_i \in S$,

2) $mod_{\mathrm{red}} \cap mod_{\mathrm{dis}} = \emptyset$,

3) $mod_{\mathrm{red}} \subseteq adm$ and $mod_{\mathrm{dis}} \subseteq adm$.

If any of the above conditions does not hold, abort.

c) Set the redacted message $m'$ to consist of the $n' = n - |mod_{\mathrm{red}}|$ message fields $m_i$ for which $i \in \{1, \ldots, n\} \setminus mod_{\mathrm{red}}$. Identify each message field by a new unique index $i' \in \{1, \ldots, n'\}$ and let $f : \{1, \ldots, n\} \setminus mod_{\mathrm{red}} \to \{1, \ldots, n'\}$ denote the induced mapping from the original index values $i$ to the new index values $i'$ (i.e. denoting the message fields of $m'$ as $m'_1, \ldots, m'_{n'}$, then $m'_{f(i)} = m_i$ for $i \in \{1, \ldots, n\} \setminus mod_{\mathrm{red}}$).

d) Set $\Sigma' = \Sigma$

e) For each $i \in mod_{red}$, retrieve signature $\sigma_i$ from $S$, and remove $\sigma_i$ from $\Sigma'$: set $\Sigma' = \Sigma' \cdot \sigma_i^{-1}$

f) Set $adm'$ to be the redacted admissible changes with respect to the new index values $i'$ : $adm' = \{ f(i) \mid i \in adm \setminus (mod_{red} \cup mod_{dis}) \}$.

g) Set $S'$ to be the set of signatures $\sigma_i \in S$ corresponding to the redacted admissible changes: $S' = \{\sigma_i \mid \sigma_i \in S, f(i) \in adm'\}$. For each $\sigma_i \in S'$, associate $\sigma_i$ with index $i' = f(i)$.

h) Set $att' = (tag, \Sigma', S')$

The process returns:

— redacted message $m'$;

— redacted attestation $att'$;

— redacted admissible changes.

The encoding of $m'$, $att'$, and $adm'$ should rely on the new index values $i' \in \{1,\ldots,n'\}$ and shall not depend on the index values $i$ used to describe the input message $m$, attestation $att$, and admissible changes $adm$.

NOTE 1 The process ensures that any message field $m_i$ for which $i \in mod_{dis}$ cannot be redacted in subsequent redactions of $m'$ and $att'$.

NOTE 2 The output of the redaction process is perfectly indistinguishable from the output of the redactable attestation process with input $m'$ and $adm'$, and removes any information regarding the redacted messages fields (which cannot be inferred from $m'$), including the number of redacted fields.

### 10.2.4 Verification Process

The verification process takes as input:

— domain parameters $Z$;

— verification key $vk$;

— message $m$ consisting of $n$ fields $m_1, \ldots, m_n$, each identified by a unique index $i$ in the range $1 \le i \le n$;

— attestation $att = (tag, \Sigma, S)$.

The process is as follows:

a) Set $o = accept$.

b) Compute the hash-code $h_0 \in G_1$ of the $tag$: $h_0 = \mathrm{H}(tag)$.

c) For each $i = 1, \ldots, n$, compute the hash-code $h_i \in G_1$ of $tag$ concatenated with the message field $m_i$: $h_i = \mathrm{H}(tag \,||\, m_i)$.

d) If $e(\Sigma, g_2) \neq \prod_{j=0}^{n} e(h_j, vk)$, set $o = reject$.

e) For each $\sigma_i \in S$, if $e(\sigma_i, g_2) \neq e(h_i, vk)$, set $o = reject$.

The process outputs:

— verification result $o$.

NOTE 1 The redactable attestation scheme specified in 10.2.1 to 10.2.4 satisfies the following properties: unforgeability, privacy, undetectability of redactions, disclosure control, and consecutive redaction control. See Annex B for further details.

NOTE 2 This scheme was originally introduced by K. Miyazaki, G. Hanaoka, and H. Imai in 2006.[11]

## 11 Scheme MIMSYTI05

### 11.1 Parameters

The MIMSYTI05 scheme makes use of the following parameters:

— digital signature scheme as defined in ISO/IEC 14888-1;

— message commitment scheme;

— security parameter $\lambda$.

The security parameter $\lambda$ shall be met by all involved algorithms, especially for the digital signature scheme and the message commitment scheme.

### 11.2 Construction

#### 11.2.1 Key generation process

The attestor generates a private/public key pair (*sk*, *pk*) in manner of the signature scheme used as a building block. The key generation algorithm of the redactable attestation scheme consists of the following two procedures:

a) generate the set of domain parameters *Z*;

b) generate the attestation key $ak$, verification key $vk$ and redaction key *rk* as follows:

   i) using the key generation process of the digital signature scheme, such that the digital signature scheme's signature key *sk* maps to the attestation key *ak*,

   ii) the digital signature scheme's verification key *pk* maps to the verification key *vk*,

   iii) the digital signature scheme's verification key *pk* also maps to the redaction key *rk*.

The security parameter $\lambda$ is taken into account, such that the digital signature scheme offers at least the indicated security strength.

The process does not output a dedicated redaction key *rk* as the redaction process of this scheme does not require it.

#### 11.2.2 Redactable attestation process

The process takes the following inputs:

— set of domain parameters *Z*;

— attestation key $ak$;

— a message consisting of *n* fields $m=(m_1,\ldots,m_n)$;

— admissible changes *adm*.

NOTE 1 The admissible changes *adm* for this scheme consist of three sets *C, DA, DB*. They represent the conditions "redacted", "Disclosed and additional redaction is allowed", and "Disclosed and additional redaction is prohibited", respectively. Their initial conditions are set to $C = \emptyset$, $DA = \{i|1 \leq i \leq n\}$, and $DB = \emptyset$.

The process Split() to split the message $m$ into fields is generally out of scope of this document. However, the process Split() shall be defined in such a way that fields can be transformed into the message and vice versa in a reproducible way everytime this conversion is necessary, i.e. if Split($m$)= $m_1,\ldots,m_s$ then Split$^{-1}$($m_1,\ldots,m_s$) = $m$. The scheme protects the content of each field.

The process requires the generation of random data. Refer to ISO/IEC 18031 to generate this securely.

The process is as follows:

a) Generate a $\lambda$-bit length random bit string $r_i$ for each field $m_i$ ($1 \le i \le n$).

b) Generate $n$ other $2\lambda$-bit length random bit strings $S_i$ ($1 \le i \le n$). The random bit string $S_i$ is referred to as the mask data for each field $m_i$. The left-most or right-most $\lambda$ bits of $S_i$ are denoted as $S_i^L$ or $S_i^R$ .

c) Calculate two values $Q_i$ and $P_i$ for each field $m_i$ such that the two points (0, $Q_i$), (3, $P_i$) are on the line $l_i$ through the two points (1, $c(R_i)$) and (2, $c(S_i)$), where $c(R_i)$ := Com($m_i$ ; $r_i$) and $c(S_i)$:= Com($S_i^L$; $S_i^R$) and Com(· ; ·) is a message commitment function. This calculation shall be done on a sufficiently large finite field with respect to the security parameter.

d) Generate a signature $\Sigma$ for the concatenation $Q_1 \| Q_2 \| \ldots \| Q_n \| P_1 \| P_2 \| \ldots \| P_n$ with the attestation key $ak$.

The process outputs:

— redactable attestation $att$=($\Sigma$, $n$, $tag$), where $tag$=($\{r_i\}_{i=1}^n$, $\{S_i\}_{i=1}^n$, $\{P_i\}_{i=1}^n$)

— redaction key $rk$, which is equal to the verification key $vk$.

NOTE 2 In practical use, a message authentication code (MAC) specified in the ISO/IEC 9797 series can be used for a message commitment function Com(· ; ·). The security parameter $\lambda$ is taken into account such that the MAC offers at least the indicated security strength.

NOTE 3 Theoretically, it is equivalent to use the concatenation $c(R_1)\|c(R_2)\| \ldots \|c(R_n)\|c(S_1)\|c(S_2)\| \ldots \|c(S_n)$ as an input to the signature scheme instead of $Q_1 \| Q_2 \| \ldots \| Q_n \| P_1 \| P_2 \| \ldots \| P_n$. In the former case, it is not necessary to recover the point $Q_i$ at verification process. This change reduces the computation cost if the field $m_i$ is not redacted and yet redactable.

### 11.2.3 Redaction process

The process takes the following inputs:

— set of domain parameters $Z$;

— a message $m$ composed of $n$ fields denoted by $m_1, \ldots, m_n$;

— attestation $att$=($\Sigma$, $n$, $tag$);

— redaction key $rk$;

— admissible changes $adm$;

— modification instructions $mod$, which are represented as a set of the unique indices $i$ that correspond to the indices of the fields that will be redacted, and are in accordance with the admissible changes $adm$.

The process is as follows:

a) Verify that $att$ = ($\Sigma$, $n$, $tag$) is a valid attestation on $m$ under the verification key $vk = rk$ and abort if this is not the case.

b) Check the modification instructions $mod$ = ($C_{\mathrm{mod}}$, $DA_{\mathrm{mod}}$, $DB_{\mathrm{mod}}$): If $C_{\mathrm{mod}}$ or $DB_{\mathrm{mod}}$ is not a subset or equal to $DA$ abort.

c) Calculate the redacted admissible changes $adm' = (C', DA', DB')$ as follows:

   i) $C' = C \cup C_{\mathrm{mod}}$

   ii) $DA' = DA_{\mathrm{mod}}$

   iii) $DB' = DB \cup DB_{\mathrm{mod}}$

d) Set $m'$ to those fields that shall remain, i.e. remove all $m_i$ if $i \in C_{\mathrm{mod}}$.

e) Calculate the redacted tag as follows: $tag' = (\{r_i\}^n_{i=1} \setminus \{r_i\}_{i \in C'}, \{S_i\}^n_{i=1} \setminus \{S_i\}_{i \in DB'}, \{P_i\}^n_{i=1})$

The process outputs:

— redactable attestation $att=(\Sigma, n, tag')$;

— redacted message $m'$;

— redaction admissible changes $adm'$.

NOTE The modification instruction can include the instruction to modify adm, e.g. by moving the index $x$ from set $DA$ to $DB$ instructs the redaction process in step d) to remove the necessary information ($S_x$) which would be required to perform a future redaction of $m_x$. This makes the scheme achieve the property of consecutive redaction control. See Annex B for further details.

### 11.2.4 Verification Process

The process takes the following inputs:

— set of domain parameters $Z$

— verification key $vk$

— message $m = (m_1, \dots, m_n)$

— attestation $att=(\Sigma, n, tag)$

— admissible changes $adm = (C, DA, DB)$

The process is as follows:

a) Set $o = accept$

b) Check if the message $m$ is consistent to the admissible changes $adm$. If it is not the case, set $o = reject$.

c) Check if the attestation $att$ is consistent to the admissible changes $adm$. If it is not the case, set $o = reject$.

d) Recover the point $(0, Q_i)$ in the following manner:

   i) If $m_i$ is not a redacted field, calculate the value $Q'_i$ such that the point $(0, Q'_i)$ is on the line $l'_i$ through the two points $(1, c(R_i))$ and $(3, P_i)$.

   ii) If $S_i$ is in a redacted field, calculate the value $Q'_i$ such that the point $(0, Q'_i)$ is on the line $l'_i$ through the two points $(2, c(S_i))$ and $(3, P_i)$.

e) Verify the correctness of $\Sigma$ by executing the digital signature scheme's verification process on inputs:

   i) the concatenation $Q'_1 \| Q'_2 \| \dots \| Q'_n \| P_1 \| P_2 \| \dots \| P_n$,

   ii) set of domain parameters $Z$,

   iii) verification key $vk$,

   iv) signature $\Sigma$,

receiving an output $o_{DSS}$.

f) If $o_{DSS}$ = *reject*, set $o$ = *reject*.

The process outputs:

— the final verification outcome $o$.

NOTE 1 The redactable signature scheme specified in 11.2.1 to 11.2.4 satisfies the following properties: unforgeability, privacy, detectability of redactions, and consecutive redaction control. See Annex B for further details.

NOTE 2 This scheme was originally introduced by K. Miyazaki, M. Iwamura, T. Matsumoto, R. Sasaki, H. Yoshiura, S. Tezuka, H. Imai in 2005.[14]

# Annex A
(normative)

# Object identifiers

This annex lists the object identifiers assigned to the mechanisms specified in this document.

```
RedactableSignatureSchemes {
     iso(1) standard(0) redaction-of-authentic-data(23264) part2(2)
        asn1-module(0) object-identifiers(0) }

DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS All; --

-- IMPORTS None; --

redactableSignatureScheme OBJECT IDENTIFIER ::= {
     iso(1) standard(0) redaction-of-authentic-data(23264) part2(2) mechanisms(1)
}

generic        OBJECT IDENTIFIER ::= { redactableSignatureScheme 1 }
sbz02mersaprod OBJECT IDENTIFIER ::= { redactableSignatureScheme 2 }
bbdffkmopps10  OBJECT IDENTIFIER ::= { redactableSignatureScheme 3 }
dpss15         OBJECT IDENTIFIER ::= { redactableSignatureScheme 4 }
mhi06          OBJECT IDENTIFIER ::= { redactableSignatureScheme 5 }
mimsyti05      OBJECT IDENTIFIER ::= { redactableSignatureScheme 6 }

END
```

# Annex B
(informative)

# Overview of properties of redactable signature schemes based on asymmetric mechanisms

## B.1 General

Table B.1 gives an overview of the security properties of all redactable signature schemes presented in this document. The properties are described and defined in ISO/IEC 23264-1. An “X” in Table B.1 indicates that the scheme fulfills this property.

All schemes presented fulfill the required mandatory cryptographic properties of correctness, unforgeability and privacy.

**Table B.1 — Overview of properties of redactable signature schemes based on asymmetric mechanisms**

| Scheme | Unforgeability | Privacy | Undetectability of redactions | Detectability of redactions | Unlinkability of redactions | Disclosure control | Consecutive redaction control | Mergeability |
|---|---|---|---|---|---|---|---|---|
| Generic construction (Clause 6) | X | X |  | X |  |  |  | X |
| SBZ02-MERSAProd[12] (Clause 7) | X | X |  | X |  | X |  |  |
| BBDFFKMOPPS10[4] (Clause 8) | X | X | X |  |  |  |  | X |
| DPSS15[5] (Clause 9) | X | X | X |  |  | X |  | X |
| MHI06[11] (Clause 10) | X | X | X |  |  | X | X | X |
| MIMSYTI05[14] (Clause 11) | X | X |  | X |  | X | X |  |
| NOTE "X" indicates that the scheme fulfills the relevant property. | | | | | | | | |

B.2 to B.7 provides background on where the cryptographic properties listed in Table B.1 have been proven.

## B.2 Properties of generic construction

The redactable attestation scheme specified in 6.2.1 to 6.2.4 satisfies the following properties: correctness, unforgeability, privacy, detectability of redactions and mergeability.

This scheme was originally introduced by R. Steinfeld, L. Bull, and Y. Zheng in 2001.[12] In that paper, the properties of correctness, unforgeability, privacy are discussed and proven. Detectability of redactions directly follows from the fact that the initial number of message fields $n$ is signed [the signature scheme signs $\left(root, tag_{msg}, n\right)$] and thus cannot be adapted during redaction. A verifier that is able to successfully verify a redacted attestation over less than $n$ fields can therefore detect the redaction and also knows the number of redacted message fields.

Mergeability follows from the underlying Merkle tree.[15] Mergeability for the generic construction can be shown as follows: let *att′* and *att″* be two valid redacted attestations on two redacted messages *m′* and *m″*, which are derived from the same original message and its redactable attestation. Assume from *m′* one has knowledge of a subordinate node's content and its tag that was redacted in *m″*. The knowledge of a node's content $m_x$ ′ and its tag $tag_{m_x}$ ′ would allow to refit the original content of the field $m_x$ into the attested message *m″* at the original position instead of the hash value and the value from $tag_{m_i}$ ′ into the list of tags in *att″* instead of the value $0^{\lambda}$. This would then generate a valid attestated message containing the fields from *m‴* and additionally the $m_x$ from *m′* at the position *x*.

## B.3 Properties of SBZ02-MERSAProd

The redactable attestation scheme specified in 7.2.1 to 7.2.4 satisfies the following properties: unforgeability, privacy, detectability of redactions, and disclosure control.

This scheme was originally introduced by R. Steinfeld, L. Bull, and Y. Zheng in 2001.[12] In that paper, the properties of unforgeability, privacy are discussed and proven.

Redactions can efficiently be detected by checking whether the value *n* contained in the redacted or redactable attestation $att=(\Sigma,n,tag_{CES})$ corresponds to the number of blocks contained in the redacted or original message $m$. As *n* is hashed into every $h_i$ [see 7.2.2. b)] it can not be adjusted during redaction.

Disclosure control allows the attestor to define the admissible changes in a way that one or more fields cannot be redacted in the redaction process; this is provided by protecting *adm* within the signature.

## B.4 Properties of BBDFFKMOPPS10

The redactable signature scheme specified in 8.2.1 to 8.2.4 satisfies the following properties: unforgeability, privacy, undetectability of redactions, and mergeability.

This scheme was originally introduced by C. Brzuska, H. Busch, O. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder in 2010.[4] In that paper, the properties of unforgeability, privacy, and undetectability of redactions are discussed and proven.

The property of mergeability was later discussed and introduced in Reference [15] as an extension of Reference [4]. Reference [15] states that Reference [4] "can be considered secure in our enhanced security model".[15] Thus, the scheme BBDFFKMOPPS10 is mergeable.

## B.5 Properties of DPSS15

The redactable signature scheme specified in 9.3.1 to 9.3.4 satisfies the following properties: unforgeability, privacy, undetectability of redactions, disclosure control and mergeability.

This scheme was originally introduced by D. Derler, H. C. Pöhls, K. Samelin, and D. Slamanig in 2015.[5] In that paper, the properties of unforgeability, privacy, undetectability of redactions, disclosure control are discussed and proven.

The original paper does not explicitly state the property of disclosure control by using the same term, however the scheme clearly allows the attestor to define the admissible changes in a way that one or more fields cannot be redacted in the redaction process; as *adm* is protected by the signature. The security proof given in Reference [5] covers an attack to change *adm*.

The property of mergeability was introduced in Reference [15] as an extension. Mergeability of DPSS15 can be shown as follows: let *att′* and *att″* be two valid redacted attestations on two redacted messages *m′* and *m″*, which are derived from the same original message and its redactable attestation. Facilitating the RSA Accumulator, the value of *acc* does not change during redaction, but a redaction removes the field and the corresponding witness value, e.g. for a redacted $m_x$ from *m′* the missing witness would successfully prohibit running the accumulator's algorithm AVerify to validate if a redacted $m_x$ was signed. However, if $m_x$ and the corresponding witness is still available in $m''$, then taking that witness and the corresponding contents of the element from $m''$ allows $m_x$ to be added into $m''$ to form a new message with a valid attestation.

## B.6 Properties of MHI06

The redactable attestation scheme specified in 10.2.1 to 10.2.4 satisfies the following properties: unforgeability, privacy, undetectability of redactions, disclosure control, consecutive redaction control and mergeability.

This scheme was originally introduced by K. Miyazaki, G. Hanaoka, and H. Imai in 2006.[11] In that paper, the properties of unforgeability, privacy, undetectability of redactions, disclosure control, consecutive redaction control are discussed and proven.

Reference [11] does not explicitly state the property of disclosure control by using the same term, however the scheme clearly allows the attestor to define the admissible changes in a way that one or more fields cannot be redacted in the redaction process; as *adm* is protected by the signature. The security proof given in Reference [11] covers an attack to change *adm*.

Reference [11] does not explicitly state the property of consecutive redaction control by using the same term but calls it “disclosure condition control”. This control is the same and enables the redactor to disallow a future redaction by moving a redactable message field into the set $mod_{dis}$.

The property of mergeability was later discussed and introduced in Reference [15] as an extension of Reference [11], which is considered secure in this enhanced security model. Thus, the scheme MHI06 is mergeable.

## B.7 Properties of MIMSYTI05

The redactable signature scheme specified in 11.2.1 to 11.2.4 satisfies the following properties: unforgeability, privacy, detectability of redactions, disclosure control, and consecutive redaction control.

This scheme was originally introduced by K. Miyazaki, M. Iwamura, T. Matsumoto, R. Sasaki, H. Yoshiura, S. Tezuka, H. Imai in 2005.[14] In that paper, the properties of unforgeability, privacy, detectability of redactions, disclosure control, and consecutive redaction control are discussed and proven.

Reference [14] does not explicitly state the property of disclosure control by that term, however the scheme clearly allows the attestor to define the admissible changes in a way that one or more fields cannot be redacted in the redaction process. This control is described as “disclosed and additional sanitizing is prohibited” and is covered in the security proof in Reference [14].

Reference [14] does not explicitly state the property of consecutive redaction control by using the same term, however the modification instruction can include the instruction to modify *adm*, e.g. by moving the index $x$ from set $DA$ to $DB$ instructs the redaction process in 11.2.3 step d) to remove the necessary information ($S_x$) which would be required to perform a future redaction of $m_x$. This enables the scheme to achieve the property of consecutive redaction control.

# Annex C
(informative)

## Criteria for inclusion of schemes in this document

Included schemes fulfil the cryptographic properties of privacy and unforgeability as defined in ISO/IEC 23264-1. Further, the criteria for inclusion require the scheme to be based on asymmetric cryptography.

The mechanisms included in this document have been selected from the large variety of such techniques which have been published and are in use. The exclusion of particular mechanisms does not imply that these techniques are insecure.

The mechanisms specified represent a small set of techniques chosen according to the following criteria (where the order of presentation of the criteria is not of significance):

— Minimum security strength: All schemes have a security strength comparable to that of state-of-the-art asymmetric signature methods. Additionally, the only known attacks against the scheme are not faster than generic attacks against the asymmetric signature methods.

— Public domain: The scheme's detailed description have been published for a minimum period of three years in the public domain.

— Provable security: Prior to inclusion, the schemes have their security proof published in peer-reviewed journals or conferences.

— Performance: Performance measurements occur on many different vectors such as bits/cycle, bits/watt, bits/security level. It is necessary to provide robust evidence that the scheme to be included offers better performance on the performance vectors that are optimized for the intended applications, than the existing schemes in this document.

— New security properties or combinations thereof: The scheme to be included is required to offer security proofs showing that it fulfils a new property or a combination of existing security properties, other than the existing schemes in this document. Robust evidence shall be provided that the scheme's new security property offers an increased value.

# Annex D
(informative)

# Numerical examples

## D.1 Generic construction numerical examples

### D.1.1 Parameters

The parameters used for the following example are as follows:

— any digital signature scheme as defined in ISO/IEC 14888-1;

— SHA3-256 is used as a hash-function (see ISO/IEC 10118-3);

— the security parameter is set to $\lambda$=128.

### D.1.2 Key generation process

The key material is generated according to the key generation process of the used digital signature scheme. Numerical examples for specific schemes can be found in the relevant standards, e.g. the ISO/IEC 14888 series.

### D.1.3 Redactable attestation process

The message to be signed is as follows: $m$ = (“This is ”, “a test message ”, “for ISO/IEC 23264-2.”)

The message blocks are thus defined as follows:

$m_1$ = “This is ”

$m_2$ = “a test message ”

$m_3$ = “for ISO/IEC 23264-2.”

NOTE The last character of $m_1$ and $m_2$ is a whitespace.

Accordingly it holds that $n$=3 and that $mod$={1,2,3}.

— A Merkle tree with 4 leaf nodes is generated as depicted in Figure D.1

— The following tags are chosen:

$tag_{msg}$ = `0x43fc5134 4c8486ea 22d4f142 9e70bfec`
$tag_1$ = `0x94bd9fbd d15b9b96 fbe6dd50 2ec9e5fa`
$tag_2$ = `0x69cd3ea8 a7124ea6 d55a5bac 71438eb4`
$tag_3$ = `0xb47ddfc7 5eb2710d 6e47ed06 15cd9574`

— The hash-codes are computed as follows:

$h_1$ = SHA3-256($tag_{msg}||m_1||tag_1$) =

```
        0xd66fb5b9 4545f8ab 8b6c449d 324714e1 0aff7f65 8f8cb2c0 144a6723
                                                                9b88f97a
```

$h_2$ =

```
        0x74911196 8fb37ead 470be653 39346bcf eb7e5c44 8ecbc65b 93a94fe0
                                                                657f72ce
```

$h_3$ =

```
        0xef170daf 2f0bd382 1aec3df4 6d4f1a43 7bb90cd5 5e1c1cab cfdd5fb0
                                                                b00ccd62
```

— The Merkle tree is initialized with the values above as well as $h_4$="" i.e. the empty string.

— The Merkle tree is computed as follows:

$h_{12}$ = SHA3-256($h_1||h_2$) =

```
        0x4733147f 2129fe21 7a602ba6 ee026cc6 21cd1765 64739652 cb5d22f1
                                                                ce0e0268
```

$h_{34}$ =

```
        0x440ea274 1f557c8f 9b695730 c39efbe0 5ce20ab0 21efcedf 67b2a6cb
                                                                4539df49
```

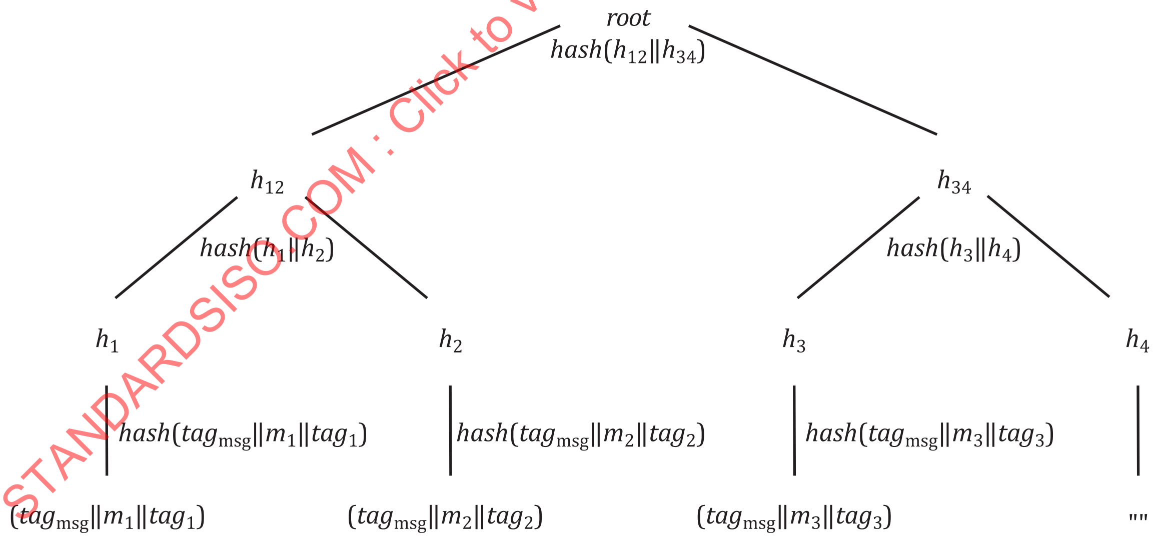
*root* =

```
        0x284f7ee7 ef4d5bc9 3e1c5cad ed05b3e6 80322260 fb4c7097 52b8e224
                                                                07cf90cc
```

— The digital signature scheme's signature process is invoked as specified in 6.1, resulting in a signature $\Sigma$.

The redactable attestation is given by *att* = ($\Sigma$,3,`0x43fc5134 4c8486ea 22d4f142 9e70bfec,(0x94bd9fbd d15b9b96 fbe6dd50 2ec9e5fa,0x69cd3ea8 a7124ea6 d55a5bac 71438eb4,0xb47ddfc7 5eb2710d 6e47ed06 15cd9574`)).

A schematic representation of the Merkle tree is given in Figure D.1.



root
hash($h_{12}||h_{34}$)
$h_{12}$
$h_{34}$
hash($h_1||h_2$)
hash($h_3||h_4$)
$h_1$
$h_2$
$h_3$
$h_4$
hash($tag_{msg}||m_1||tag_1$)
hash($tag_{msg}||m_2||tag_2$)
hash($tag_{msg}||m_3||tag_3$)
($tag_{msg}||m_1||tag_1$)
($tag_{msg}||m_2||tag_2$)
($tag_{msg}||m_3||tag_3$)
""

**Figure D.1 — Numerical example — tree before redaction**

## D.1.4 Redaction process

The redaction process takes as input the domain parameters, *att*, the message fields $m_1$, $m_2$, $m_3$, a redaction key *rk*, *adm*={1,2,3}, and *mod*=3. The latter is to indicate to redact $m_3$ from the message.

— The Merkle tree is computed in full analogy to the redactable attestation process above, and finally the verification process of the digital signature is invoked.

— The message is modified to

$m'$ = ("This is ", "a test message ", `0xef170daf 2f0bd382 1aec3df4 6d4f1a43 7bb90cd5 5e1c1cab cfdd5fb0 b00ccd62`)

— Furthermore, the attestation is modified to

$att'$ = ($\Sigma$,3,`0x43fc5134 4c8486ea 22d4f142 9e70bfec`, (`0x94bd9fbd d15b9b96 fbe6dd50 2ec9e5fa`, `69cd3ea8 a7124ea6 d55a5bac 71438eb4`, `0x00000000 00000000 00000000 00000000`))

And the admissible changes are modified to $adm'$ = {1,2}.

## D.1.5 Verification process

The verification process takes as input $m$, $att$, $adm$, $vk$, and $Z$ as output by the redaction process. Then, the verification algorithm proceeds as follows:

— After reconstructing $n$=3 from $att$, a Merkle tree with four leaf nodes is initialized.

— The leaf nodes $h_i$ are computed as follows:

— As $tag_1 \neq 0^\lambda$ and $tag_2 \neq 0^\lambda$, the process computes:

$tag_1$ = `0x94bd9fbd d15b9b96 fbe6dd50 2ec9e5fa`
$tag_2$ = `0x69cd3ea8 a7124ea6 d55a5bac 71438eb4`
$h_1$ = SHA3-256($tag_{msg}||m_1||tag_1$) =
```
0xd66fb5b9 4545f8ab 8b6c449d 324714e1 0aff7f65 8f8cb2c0
                                      144a6723 9b88f97a
```
$h_2$ = SHA3-256($tag_{msg}||m_2||tag_2$) =
```
0x74911196 8fb37ead 470be653 39346bcf eb7e5c44 8ecbc65b
                                      93a94fe0 657f72ce
```

— As $tag_3 = 0^\lambda$ the process computes:

$h_3$ =
```
0xef170daf 2f0bd382 1aec3df4 6d4f1a43 7bb90cd5 5e1c1cab
                                      cfdd5fb0 b00ccd62
```

— As $n$=3, $h_4$ is defined as the empty string.

$h_4$ = ""

A schematic representation of the Merkle tree after the redaction of $m_3$ is given in Figure D.2.

The root of Merkle Tree is now computed in analogy to the redactable attestation process. Finally, the verification process of the digital signature scheme is invoked on the inputs specified in 6.1.

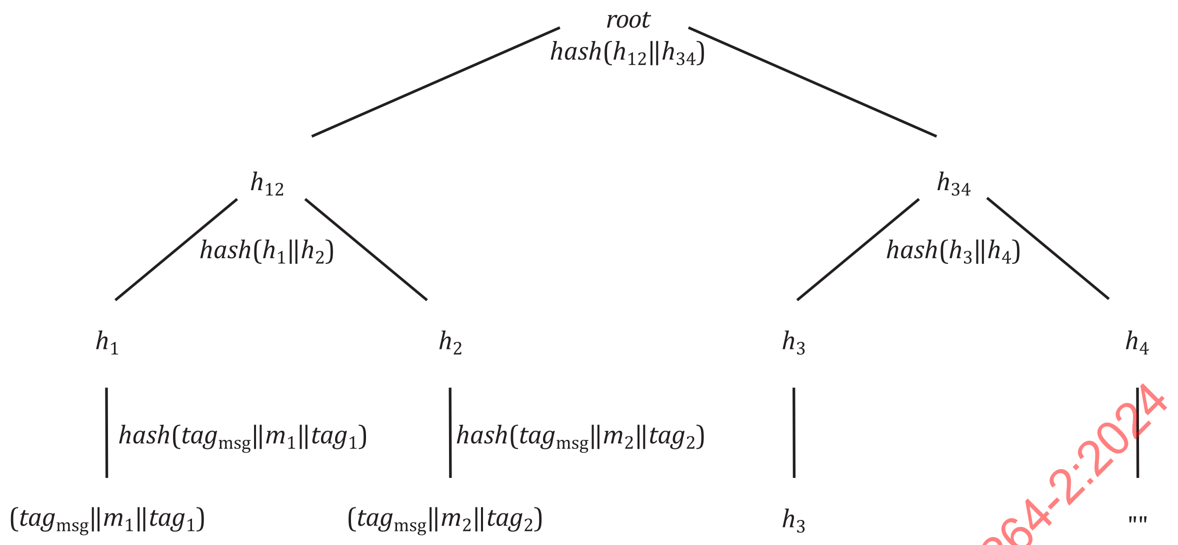

$root$
$hash(h_{12}||h_{34})$
$h_{12}$
$hash(h_1||h_2)$
$h_{34}$
$hash(h_3||h_4)$
$h_1$
$h_2$
$h_3$
$h_4$
$hash(tag_{msg}||m_1||tag_1)$
$hash(tag_{msg}||m_2||tag_2)$
$(tag_{msg}||m_1||tag_1)$
$(tag_{msg}||m_2||tag_2)$
$h_3$
""

**Figure D.2 — Numerical example — tree after redaction**

## D.2 SBZ02-MERSAProd numerical examples

### D.2.1 Parameters

The parameters used for the example specified in D.2.2 are as follows:

— SHA3-256 is used as a hash-function, (see ISO/IEC 10118-3).

— The security parameter is set to $\lambda$=128.

### D.2.2 Key generation process

The key material is generated according to the key generation process of the used digital signature scheme (keySize = 256).

$l$ = 3

$p$ =

```
0xc250f6e7 24c2be12 6aabdef3 2666a49a 24f26225 539f07cf 756ba3be d023ab25
  4bec5d35 cc8d9afc 5eeded6e 90e82b97 7f45d675 4ee8c4df b0307935 494783f9
  c4e275e6 0690e7ac 178d445b 7d854c67 00e33f0b 2adf3b23 ea10d92d f1ee33c6
  0bca7796 2734c021 2c24837f 25938fee 21d8c949 d429a17d cbb1adb1 50a8775b
```

$q$ =

```
0xab99788d 93065b0e 27fcc3ee 97a46a71 db4a0a8e 093e4087 906b3614 24f855ea
  ae0b2573 0d541696 9d20ae45 c06c0c34 fd8ced94 0436ef72 d03cb401 8997c5f1
  19c3b442 c3f8b453 30ab80ff 714d37d7 cff1dc8b bd85a566 19c92a9d 90d4e939
  6c97f327 19ffc57a 230a44b9 2c7af77c 8ffb2dc3 b6f654e0 aaf40ba7 07a5462b
```

$N = p \cdot q =$

```
0x824092d1 5fa0b4ae 78b06d3d 479e700a 4249dcc9 aea610c2 6ad12f8d a1d28cc3
  5d0e1c36 061b7590 f0f9af9a 926c5e4c 303c645d a3627f11 30a1605a 73851868
  8f30655e 6e9134d0 6869f366 1de7a418 aa6ba129 5b922a40 ded552f7 53259937
  46583306 0d4357ef e82014f4 18ec56e2 a22fa352 e7465ec0 78cb92e8 ec54558c
  72e72db7 aa8169c5 609eff1e 1215b1ac 44677ea3 ee63498b 7f76a6d7 192984d2
  16546363 96a37a59 069df919 2b064c0a 18f94f52 00e29c0f 0a9d57b7 747b71c3
  6f8851c6 ccb45805 b6e2142f d7c81b4c 9ad22788 d2d4217c 61324451 1a93c1e6
  c351738d 10730ed3 7bb565ed 00e549fa c6d21ed7 99c0a3af a497060c 0795ee49
```

$e_1$ = 0x10001
$e_2$ = 0x10003
$e_3$ = 0x10007

$phi = (p-1)\cdot(q-1)$ =

```
0x824092d1 5fa0b4ae 78b06d3d 479e700a 4249dcc9 aea610c2 6ad12f8d a1d28cc3
  5d0e1c36 061b7590 f0f9af9a 926c5e4c 303c645d a3627f11 30a1605a 73851868
  8f30655e 6e9134d0 6869f366 1de7a418 aa6ba129 5b922a40 ded552f7 53259937
  46583306 0d4357ef e82014f4 18ec56e2 a22fa352 e7465ec0 78cb92e8 ec54558b
  04fcbe42 f2b850a4 cdf65c3c 540aa2a0 442b11f0 91860134 799fcd04 240d83c2
  1c5ce0ba bcc1c8c6 0a8f5d64 d9b2143d 9c268b48 adc2e7bc 8a302a80 a19c27d8
  90e2279e 022abc06 6ea94ed4 e8f5970d c9fd0bf1 ea6f40f2 5d584085 97d0a4e7
  4aef08cf cf3e8938 2c869db4 aed6c290 14fe27ca 0ea0ad51 2df14cb3 af4830c4
```

$d_1 = e_1^{-1}$ modulo $phi$ =

```
0x18b239a4 2d27815a 9b538842 d8337836 d9bebb78 19ff2eaa 4e2a7f83 b12d04b7
  7e0b50da 056a6b6e 21e295a3 0f2430f4 ed286637 42e54806 705c08b2 ddb85c9a
  669bdc7c 7d265707 34e4f1db 68cc24d6 f0691bd3 85a33f13 b1ac6ce0 ac0932f5
  c206b315 f0a9d620 8cb89880 77c147f0 0ca44fa9 4664974e 9318b630 27af0dd1
  200ef3dc 29bd3cbd 22b81560 f2e65953 5dd974bb 523c403a 3cdd49b8 0145f6b7
  753d3ef6 e9172d04 2fb86312 de1bdfae 96ced015 e5ac50d7 f4b3913b 7668ef3d
  26ab5aab a0e98d61 02d66336 073d9978 69d5eacf e18d5a99 198dc41e 105ce507
  5db3d8cd de13935c ed4450de 934ff0ef f4ab03cc 51ddb13e f5210d53 fc6609a9
```

$d_2$ =

```
0x325b5431 8a7e48a0 62b43dff 69445bd0 43f436ca 2d719842 01fa5e8d 8c99fbb2
  4d515ea0 1f381c29 f74d0327 9f899b50 6eecec9f 16200074 aa09465e e57112fe
  a3f0703f d3cc14a1 1d9b2dd3 3fb221f1 4b7fbe4d fc876b53 d6abc212 6d244ff1
  d912ea71 65a88a74 17c8afff d7eb484d c27f578d 34d72c78 b0c9d2ad 29e79564
  017f9b28 9cd1c068 968fae36 2073c348 e0c06cac 569e6578 1365b858 a2ce7937
  8ec4946d 3556e415 3531922d 194b77c4 f2a2670e a2526615 48c588fe 24812c3c
  4c3f2b47 ab4b77d3 ad29da06 ef04b69a a6fa2ac1 c5b83659 915e3cc7 703dd028
  2416428b 60a431ac ece03d34 75c392ea 6ffe7b72 c2938387 ca45de30 16028d23
```

$d_3$ =

```
0x17d869b6 9f92d071 0a938b47 5c4c550f 1d6f35f9 174156e2 285d8835 25ffc1b5
  6cf09dc7 f7b07b38 16062ce2 83c36aff e8098661 9441f7d3 ffea5c7a a831f1f5
  6f1963ad 8b61984f 4a22f712 9dd15b8a df4faf89 04bbe6c7 271a5f03 1d4c6231
  b66e22fc 93bf9d39 691f17dc 7614f2e9 c42b79a7 d72f3908 9021dccd c9bc80ad
  fa3ceaaf 59231d16 10f7376a abbc6c55 7f84a21d 6be6e4a2 3a211222 d621fa84
  cd8dc668 e3b704e0 58468bbc a7cf0b19 66082861 48df644f ffe10870 e8007e2c
  2c79a0cb 817706c1 c2308252 5d128391 76542f45 94f421e8 328bc9a5 b21edd89
  26312e75 bf0a8ad7 d4082d60 9a2aacde a5317030 e64d50fe 5903f1de 7cef3e1b
```

### D.2.3 Redactable attestation process

The message to be signed is as follows:

$m$ = (“This is an example for ”, “the MERSAProd scheme as defined in ”, “ISO/IEC 23264-2.”)

It is important to note that the scheme does not protect a particular order of the message blocks.

$m_1$ = “This is an example for ”
$m_2$ = “the MERSAProd scheme as defined in ”
$m_3$ = “ISO/IEC 23264-2.”

NOTE The last character of $m_1$ and $m_2$ is a whitespace.

The admissible changes are represented as a bit mask where the bit at the position $i$ is equal to 1, if the message block at this position is redactable, and equal to 0 otherwise.

All message blocks indicated in the following bit mask are marked as redactable, where the *i*-th least significant bit is set to `1` if $m_i$ is redactable, and to `0` otherwise:

$adm_{\mathrm{red}}$ = `0b110` = `0x06`

The complementary bit mask results for $adm_{\mathrm{fix}}$ = `0b001` = `0x01`.

*adm* is thus defined as *adm*=`0x06`||`0x01`.

There are three message blocks: *n* = `0x03`.

The following tag for the complete message is chosen uniformly at random:

$tag_{\mathrm{CES}}$ = `0x840962b0 d322743f c1909957 5894ebab`

The tags are the following (it should be noted that tag $h_i$ is the tag for the message with index *i* and $m_i$ is the message with the index *i*):

$h_1$ = SHA3-256($adm||tag_{\mathrm{CES}}||n||1||m_1$)
  = SHA3-256(`0x06`‖`0x01`‖`0x840962b0d322743fc19099575894ebab0x03`‖`0x03`‖`0x01`‖"This is an example for "), and similarly for the subsequent indices:

$h_1$ =
```
0x4e0b354b 48c91c17 c8ab9441 634bcd6a 83b75308 b28a27b6 f6548081 dd7ae6da
```

$h_2$ =
```
0x28299230 050ba6d4 ee071a0d 1eba34eb a8ecc40c e39b141f 29cca27b 55050e13
```

$h_3$ =
```
0x65685e94 d8f68516 5d7e2ffb 505992f5 42fdac6a 4e173d20 5b28168f 8e7ead0c
```

The signatures per field $s_i$ are are obtained as:

$s_1 = h_1^{d_1}$ modulo $N$ =
```
0x5565afc3 03fa91a7 844abd4c 9cb6ce98 db6a5811 3ae1e9d3 ce965d14 bf155778
  75b12f8a 585d92b6 cc2f384e ee656f4f bbe14ff8 0fd262ea 833757f5 fe222492
  a41e8f59 d3ba8a36 f12ae1c2 ca680e69 18ca3ba2 d0d93562 d2c9f3a3 1295ea3e
  f482d314 d2607e70 6bb86925 439dea53 0eedceda 97600740 4c0c7716 0e7d8e01
  ef8d3be5 6e976a42 bc23601b b33cb967 8cba2b27 4bb286d3 e0c8bf72 6cf24dbd
  1f68b9f5 5aedcd51 224e181d 958dc229 af5ae9b5 bdf935b7 ddd8d20c 897e1178
  15d13a46 db0d37b7 55f52858 867722e9 81b0c7f9 f2f679b8 08fa056b 3eb3d76e
  4bc2c517 cf568f22 4f906eda a504e11c 06c21379 4a10c450 3d46276f d6eac7b7
```

$s_2$ =
```
0x2ead4d08 71a3d827 935ec0a8 ca359ecd 4a091511 931bee06 19e90192 c14d6f5f
  5649df6d 0d8d7218 f152261f 70255416 59fc231e d3fd2654 0ab4fb03 8270ce96
  a434f50e b004dda9 53aa1c68 81467e9b f534c6de 6a960d3d 115fa1b3 f12363f3
  cbb1b8d7 5e1552dd e9d96c3f e83cad8d 4acb9ca6 a2240cc8 38de7f14 85bb777f
  29522df1 2560ae8d f6c43d2d c830284c 3adf89bb b3db7ca7 e1eeafc9 342725b5
  300a256b 20a2d2b3 8c2fd35e e496c1c9 743c6fe6 3d48326a f91e3119 9aa50b7d
  317c1f33 7e690647 9de4b7b9 ea90b1d3 827cc775 1be3608c 19b7f855 ce66b640
  2833fb5a a92fad8e 9f9f027d 9292faae 4f487114 65752121 5af6c2dd 02bf21f5
```

$s_3$ =
```
0x1325eb0a c368a709 73c54db9 3d84295f 26e9f89c bb9b1f0d f2147c92 b61d73df
  0ad1465c 0ef519cc 500c5868 00a804a1 2bdb7d32 bae86afa 281d546f 741cad07
  9088a003 152401a2 778edbea 2a6e7cba 38db4c49 14eb1ffb 667b6cd6 acbfccf9
  6c47a077 50df985c 6d45f767 c03be708 f50b4fdc c0dab4f1 6482b52c 17485a94
  a243cc21 e6a26d96 069228dc 2b75e90c 85e29129 03af60de 04d90d31 2b661a2d
  d1f6b469 22f6c20b 019af52d 3fe67110 6bf2b2f9 28351ffa 97de9265 54457e1c
  1c5f440a 5995ef17 0e3713e8 195006a9 e1c6f92d b010a4d7 57e934bd fd7f9d1c
  751cbc57 45bf3c77 3e4892ec 0fd9bbd8 edc35b61 fb0dc047 54154f47 8a34d5fe
```

The resulting $\Sigma$ is now given by the product of the individual signatures:
$\Sigma = s_1 \cdot s_2 s_3$ modulo $N$ =

```
0x333eed17 ebc7f517 ecbfcc76 49731986 6e28bd5c 3fde8300 5339b69f b4bbce5a
  259b4bbb 64442118 9769f7bf 2ed35198 639fc7c1 6960873e 0c480bd8 4eb7acf5
  b8570660 d72d48cc 8b4fc0cd b69c8f0b 37a4fc67 344b5ac8 74391146 ef2b5cea
  dfa301a6 624a7203 0751dc6d 1735fcff 85a8dec0 30d4cb3f 017ded06 3772a828
  426ddf7d 076c43fd f83c51ea 5ce0df01 d636e3cb 29774712 3de8b9ff f0d016ea
  c73610b2 6c74f606 d02e863e 79d93275 a95bc4e4 0212a8b0 a54d710e cc80ecd2
  6ad45a7f e9e19cae 2e2750a3 b56de99b d6e7214b cb4e00bd 9719f89f 422a2c3e
  23edd9fb bba46330 ab108c0f db10686b b9bbf9f7 20c6ddf5 a8581cbc ef4f5216
```

The attestation is now given as $att = (\Sigma, n, tag_{\mathrm{CES}})$.

### D.2.4 Redaction process

The input are the domain parameters, *att*, the message fields $m_1, m_2, m_3$, a redaction key $rk$, *adm*=`0b110`||`0b001`, and *mod*={2}. The latter is to indicate to redact $m_2$ from the message.

In order to redact the second message field (i.e. *mod* = {2}), first the specified consistency checks are performed. As the bit at the position 2 of $adm_{\mathrm{red}}$ ($adm_{\mathrm{red}}$=`0b110`) is 1 and 0 in $adm_{\mathrm{fix}}$ respectively, this message field is redactable. After redaction, the resulting message shall look like follows: $m' = \{ m_1, m_3 \}$.

For $X = \{1,3\}$, subsequently the required systems of modular equations are solved, resulting in:

$c_1$ = `0x955b d568400e`
$c_3$ = `0x0aab 2aac2001`

Recomputing the hash values results in the same values as above, which are thus omitted here.

Computing the $s_k$ according to the specification yields the original values also generated in the following values:

$$s_1 = \left( \sum^{c_1} \right) \cdot \left( h_1^{\left\lfloor \frac{c_1}{e_1} \right\rfloor} \right) \cdot \left( h_2^{\left\lfloor \frac{c_1}{e_2} \right\rfloor} \right) \left( h_3^{\left\lfloor \frac{c_1}{e_3} \right\rfloor} \right) \text{ modulo } N =$$

```
0x5565afc3 03fa91a7 844abd4c 9cb6ce98 db6a5811 3ae1e9d3 ce965d14 bf155778
  75b12f8a 585d92b6 cc2f384e ee656f4f bbe14ff8 0fd262ea 833757f5 fe222492
  a41e8f59 d3ba8a36 f12ae1c2 ca680e69 18ca3ba2 d0d93562 d2c9f3a3 1295ea3e
  f482d314 d2607e70 6bb86925 439dea53 0eedceda 97600740 4c0c7716 0e7d8e01
  ef8d3be5 6e976a42 bc23601b b33cb967 8cba2b27 4bb286d3 e0c8bf72 6cf24dbd
  1f68b9f5 5aedcd51 224e181d 958dc229 af5ae9b5 bdf935b7 ddd8d20c 897e1178
  15d13a46 db0d37b7 55f52858 867722e9 81b0c7f9 f2f679b8 08fa056b 3eb3d76e
  4bc2c517 cf568f22 4f906eda a504e11c 06c21379 4a10c450 3d46276f d6eac7b7
```

Similarly, computing $s_3$ yields the same result as in the redactable attestation process:

$s_3$ =

```
0x1325eb0a c368a709 73c54db9 3d84295f 26e9f89c bb9b1f0d f2147c92 b61d73df
  0ad1465c 0ef519cc 500c5868 00a804a1 2bdb7d32 bae86afa 281d546f 741cad07
  9088a003 152401a2 778edbea 2a6e7cba 38db4c49 14eb1ffb 667b6cd6 acbfccf9
  6c47a077 50df985c 6d45f767 c03be708 f50b4fdc c0dab4f1 6482b52c 17485a94
  a243cc21 e6a26d96 069228dc 2b75e90c 85e29129 03af60de 04d90d31 2b661a2d
  d1f6b469 22f6c20b 019af52d 3fe67110 6bf2b2f9 28351ffa 97de9265 54457e1c
  1c5f440a 5995ef17 0e3713e8 195006a9 e1c6f92d b010a4d7 57e934bd fd7f9d1c
  751cbc57 45bf3c77 3e4892ec 0fd9bbd8 edc35b61 fb0dc047 54154f47 8a34d5fe
```

The resulting redacted signature is now given by the product of these values $\Sigma' = (s_1 * s_3)$ modulo $N$ =

```
0x0825d32b e492317e 7b6cb5b7 c03f051c ef43076b ba6e0f90 de92f071 ebc4c485
  ff3c3598 eba3bd48 a1bf18ad 78aa0843 5f8ecbe3 9f4d93b4 a6fa555e b8916675
  841a7e85 0b756250 68666ab7 3379707a 092a3b5c a0af2fa4 8b29c545 6c14cd59
  0de2d331 e4480773 c0a7fbf6 589da302 d07738eb bb7875e7 893a3797 bed2eb14
  20c71a51 e3b1045e b40627b4 ce7e3e32 e51b5976 c3cb3fad bf6b6949 f774a0a6
  1e0d8c24 96442efa 1385107c d20f06fc ed7bb483 6cb2a987 87941bec 3c6fa9da
  071f21b2 faf8f014 73ed512c 2ac86fdf e60d4cf9 96be7704 cff4250c 33f18f94
  1fd8ddd7 e4b2e8ec 92fabbb3 a6b54c72 834e3120 fc6293f7 c95b8a9e c9f8114e
```

### D.2.5 Verification process

On input *m*, *att*, *adm*, *vk* , and *Z* as output by the redaction process, the verification algorithm proceeds as follows:

The input *m* consists of
$m_1$ = "This is an example for "
$m_2$ = "ISO/IEC 23264-2."

In the first steps of the verification process, the present message blocks are checked for consistency with the admissible changes (i.e. in particular whether no non-redactable message blocks have been removed). As in this example, the first and second message block were admissible for redaction, thus this check is trivially passed. Subsequently, the hash values for all non-redacted message blocks are recomputed, resulting in the same values $h_1$ and $h_2$ as before.

Furthermore, the following is obtained:
$e = e_1 \cdot e_3 =$ `0x00000001 00080007`
as well as:

$$r = h_1^{\left(\frac{e}{e_1}\right)} \cdot h_3^{\left(\frac{e}{e_3}\right)} \text{ modulo } N =$$

```
0x2cd3a518 54f54023 8b9fb1a4 c41190fb 49814b0a c3931fd2 db89f2f6 05b3c0c8
  942e3c56 1f38003e 702730eb ccc30529 91c0f365 7249083b 92dc1bcf 5d1ab0e4
  fa369c38 64961788 c33bdeef a30258ff 3845c676 bc503770 865b505c a080f4d7
  6ee4ef7b fbc36c89 d95e68c3 1ac7fe8d 23559a18 51e4c640 b80816c0 8f6a156b
  ff927593 fa84aafa 0879aafa 36f40ce2 22ff9322 c9f01ce0 e20460bb 6fdfd1ee
  419bce43 8830edf4 a115a1db 2c7c3f6a 132d129c 8575c30d 47b8e6b4 ee67c737
  5cbe0be1 e871a917 94e08d7d fc6d9f52 7227f15e 5dfedb6e 5f6fa6a7 1acb7aac
  546845c3 8cf5c928 5cca9e53 18a18297 6c264044 d2285a91 0ad737f6 95118cc2
```

Verifying that the above value *r* is identical to $\Sigma^e$ modulo *N* finally yields *o* = *accept*.

## D.3 BBDFFKMOPPS10 numerical examples

### D.3.1 Parameters

As defined in the scheme specification, the scheme makes use of the following parameters:

— any digital signature scheme as defined in ISO/IEC 14888-1;

— the security parameter $\lambda$ is set to 128.

### D.3.2 Key generation process

The key material is generated according to the key generation process of the used digital signature scheme. Numerical examples for specific schemes can be found in the relevant standards, e.g. the ISO/IEC 14888 series.

### D.3.3 Redactable attestation process

The message to be attested is defined as follows:

$m$ = (“This ”, “is a ”, “test ”, “message ” , “for ISO/IEC 23264-2.”)

The message blocks are thus defined as follows:

$m_1$ =“This ”

$m_2$ = “is a ”

$m_3$ = “test ”

$m_4$ = “message ”

$m_5$ = “for ISO/IEC 23264-2.”

NOTE The last character of $m_1$, $m_2$, $m_3$, and $m_4$ is a whitespace.

This numerical example uses the tree structure depicted in Figure D.3, but any other tree structure is possible. The tree structure is required to have a number of nodes sufficient for the number of message blocks. Here, 5 nodes and 1 root node for a message with 5 message blocks. The potential redactions depend on the tree structure as only a leaf and the associated content of that leaf can be redacted. The indices of the vertices $v_1$ , … , $v_6$ are obtained by traversing the tree structure using a post-order traversal algorithm.

The numerical example sets the value *root* , which gets assigned to the content of the node forming the root of a tree, to the string “root”. In the context of this numerical example, this uniquely marks the root node.

The contents of the vertices are set to the contents of the message blocks, i.e. $v_1$ = “This ”.

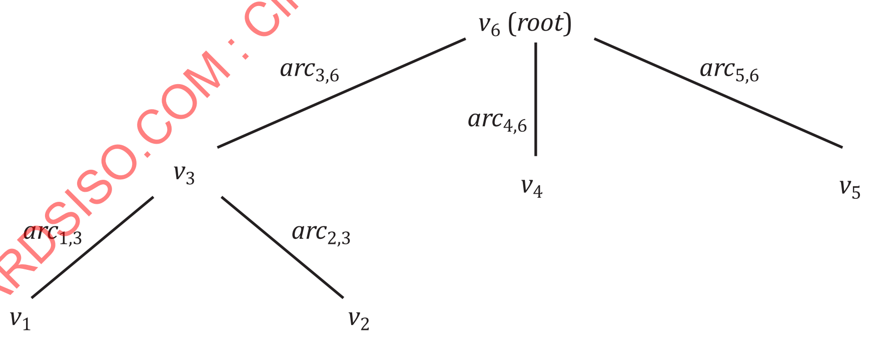


**Figure D.3 — Numerical example — tree structure example for BBDFFKMOPPS10**

For each vertex, a random bit string of length 128 is generated (leading zeros are prepended to make all tags 128 bit long in hex):

$tag_1$ = `0x07124e66 40a15927 e74825f2 167cc5d2`

$tag_2$ = `0x00007807 faf8cc83 a905eba0 3b621311`

$tag_3$ = `0xb6d1ed99 f31f4a31 11d2124a 5e457798`

$tag_4$ = `0x000ae3cb 687dbf75 b56155b7 080a7c53`

$tag_5$ = `0x0eed89ca b4b107f9 c620d0a1 5c29f4a5`

$tag_6$ = `0x0008a21c 3d791a70 7babe045 759a75e1`

For the arcs, this results in the following values being submitted to the signing process of the digital signature scheme of choice:

$m_{(1,3)} = 0 \| \text{"This"} \| tag_1 \| \text{"test"} \| tag_3$

$m_{(2,3)} = 0 \| \text{"is a"} \| tag_2 \| \text{"test"} \| tag_3$

$m_{(3,6)} = 0 \| \text{"test"} \| tag_3 \| \text{"root"} \| tag_6$

$m_{(4,6)} = 0 \| \text{"message"} \| tag_4 \| \text{"root"} \| tag_6$

$m_{(5,6)} = 0 \| \text{"for ISO/IEC 23264-2."} \| tag_5 \| \text{"root"} \| tag_6$

This generates five digital signature values which capture the contents of the vertices, thus the contents of our message. Regarding the indices, $\Sigma_{1,3}$ is the signature over $m_{(1,3)}$ , $\Sigma_{2,3}$ is the signature over $m_{(2,3)}$ and finally $\Sigma_{5,6}$ is the signature over $m_{(5,6)}$ .

Next, signatures that protect the ordering of subordinate vertices, thus the ordering of the message blocks, are generated. The following values are submitted to the signing process of the digital signature scheme of choice:

$m_{(1,2)} = 1 \| \text{"This"} \| tag_1 \| \text{"is a"} \| tag_2$

$m_{(3,4)} = 1 \| \text{"test"} \| tag_3 \| \text{"message"} \| tag_4$

$m_{(3,5)} = 1 \| \text{"test"} \| tag_3 \| \text{" for ISO/IEC 23264-2."} \| tag_5$

$m_{(4,5)} = 1 \| \text{"message"} \| tag_4 \| \text{" for ISO/IEC 23264-2."} \| tag_5$

This generates four digital signature values $\Sigma_{4,5} \| \Sigma_{3,5} \| \Sigma_{3,4} \| \Sigma_{1,2}$. For example, $\Sigma_{3,4}$ is the signature over $m_{(3,4)}$, protecting that $m_3$ with the contents “test ” is before $m_4$ with the contents “message ”. These four digital signature values are added to the front of the previous five signature values to form the redactable attestation *att* containing nine values, as follows:

$att = \Sigma_{4,5} \| \Sigma_{3,5} \| \Sigma_{3,4} \| \Sigma_{1,2} \| \Sigma_{5,6} \| \Sigma_{4,6} \| \Sigma_{3,6} \| \Sigma_{2,3} \| \Sigma_{1,3}$

Then $\Sigma_r$ is calculated over $m_r = \text{"root"} \| tag_6$ and added to the front of the redactable attestation *att* and the tags are appended at the end to form the final redactable attestation *att* containing 16 values as follows:

$att = \Sigma_r \| \Sigma_{4,5} \| \Sigma_{3,5} \| \Sigma_{3,4} \| \Sigma_{1,2} \| \Sigma_{5,6} \| \Sigma_{4,6} \| \Sigma_{3,6} \| \Sigma_{2,3} \| \Sigma_{1,3} \| tag_1 \| tag_2 \| tag_3 \| tag_4 \| tag_5 \| tag_6$

### D.3.4 Redaction process

To redact $m_4$ with the contents “message “, the signatures containing information related to $m_4$ as well as the tag $tag_4$ are removed from the values in *att*, i.e. remove $\Sigma_{4,5}$, $\Sigma_{3,4}$ and $\Sigma_{4,6}$. This means that after redaction, the verifier will have a tree structure that contains one node less, as depicted in Figure D.4. It is protected by the following redacted attestation:

$att' = \Sigma_r \| \Sigma_{3,5} \| \Sigma_{1,2} \| \Sigma_{5,6} \| \Sigma_{3,6} \| \Sigma_{2,3} \| \Sigma_{1,3} \| tag_1 \| tag_2 \| tag_3 \| tag_5 \| tag_6$

The indices used above would be different when traversing the tree from Figure D.4. However, the order will be the same when following the same traversal algorithm, and therefore the contents of the vertices and the arcs are also the same. Thus, the signatures contained in *att* will verify if the tree structure, its ordering and the contents of the vertices, i.e. the message contents, stay the same.

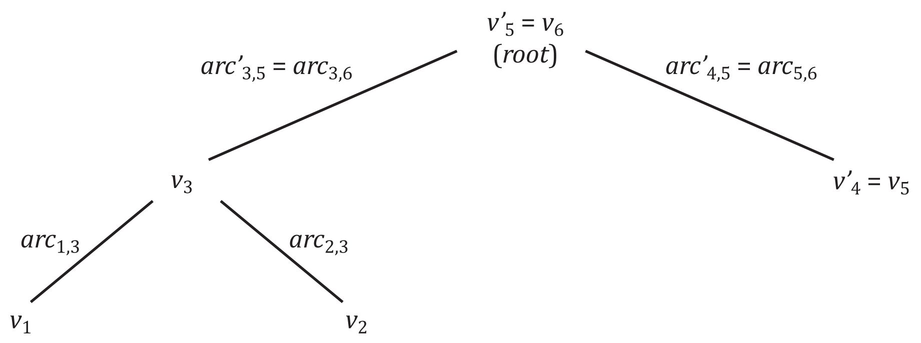


**Figure D.4 — Numerical example — tree structure after redaction of $m_4$ example for BBDFFKMOPPS10**

### D.3.5 Verification process

During the verification process, the tree structure is traversed using the same traversal algorithm as when signing, thus the ordering of the vertices and the ordering of the arcs is the same. If the contents of the vertices which contain the message blocks' contents are not changed, and the tree structure and the ordering of the vertices and thus the contents of the arcs is not changed, then the signature values contained in *att* will be the same and the signature verification process of the chosen signature algorithm will be able to verify them.

## D.4 DPSS15 numerical examples

### D.4.1 Parameters

As defined in the scheme specification, the scheme makes use of the following parameters:

— Any digital signature scheme as defined in ISO/IEC 14888-1

— For the purpose of this example, the security parameter $\lambda$ is set to $\lambda = 128$.

— As a hash-function, $\text{Hash}(x) = 2 \cdot \text{Hash}'(x) + 1$ is used, where Hash' is instantiated by SHA3-256, (see ISO/IEC 10118-3).

### D.4.2 Key generation process

The key material is generated as follows:

$(sk_{DSS}, pk_{DSS})$ are generated according to the key generation process of the used digital signature schemes.

The key material for the first accumulator is computed as follows:

$p' =$

```
0xc250f6e7 24c2be12 6aabdef3 2666a49a 24f26225 539f07cf 756ba3be d023ab25
  4bec5d35 cc8d9afc 5eeded6e 90e82b97 7f45d675 4ee8c4df b0307935 494783f9
  c4e275e6 0690e7ac 178d445b 7d854c67 00e33f0b 2adf3b23 ea10d92d f1ee33c6
  0bca7796 2734c021 2c24837f 25938fee 21d8c949 d429a17d cbb1adb1 50a8775b
```

$q' =$

```
0xab99788d 93065b0e 27fcc3ee 97a46a71 db4a0a8e 093e4087 906b3614 24f855ea
  ae0b2573 0d541696 9d20ae45 c06c0c34 fd8ced94 0436ef72 d03cb401 8997c5f1
  19c3b442 c3f8b453 30ab80ff 714d37d7 cff1dc8b bd85a566 19c92a9d 90d4e939
  6c97f327 19ffc57a 230a44b9 2c7af77c 8ffb2dc3 b6f654e0 aaf40ba7 07a5462b
```

This results in:

$sk'_{ACC} = \varphi = (p' - 1) \cdot (q' - 1) =$

```
0x824092d1 5fa0b4ae 78b06d3d 479e700a 4249dcc9 aea610c2 6ad12f8d a1d28cc3
  5d0e1c36 061b7590 f0f9af9a 926c5e4c 303c645d a3627f11 30a1605a 73851868
  8f30655e 6e9134d0 6869f366 1de7a418 aa6ba129 5b922a40 ded552f7 53259937
  46583306 0d4357ef e82014f4 18ec56e2 a22fa352 e7465ec0 78cb92e8 ec54558b
  04fcbe42 f2b850a4 cdf65c3c 540aa2a0 442b11f0 91860134 799fcd04 240d83c2
  1c5ce0ba bcc1c8c6 0a8f5d64 d9b2143d 9c268b48 adc2e7bc 8a302a80 a19c27d8
  90e2279e 022abc06 6ea94ed4 e8f5970d c9fd0bf1 ea6f40f2 5d584085 97d0a4e7
  4aef08cf cf3e8938 2c869db4 aed6c290 14fe27ca 0ea0ad51 2df14cb3 af4830c4
```

$pk'_{ACC} = p' \cdot q' =$

```
0x824092d1 5fa0b4ae 78b06d3d 479e700a 4249dcc9 aea610c2 6ad12f8d a1d28cc3
  5d0e1c36 061b7590 f0f9af9a 926c5e4c 303c645d a3627f11 30a1605a 73851868
  8f30655e 6e9134d0 6869f366 1de7a418 aa6ba129 5b922a40 ded552f7 53259937
  46583306 0d4357ef e82014f4 18ec56e2 a22fa352 e7465ec0 78cb92e8 ec54558c
  72e72db7 aa8169c5 609eff1e 1215b1ac 44677ea3 ee63498b 7f76a6d7 192984d2
  16546363 96a37a59 069df919 2b064c0a 18f94f52 00e29c0f 0a9d57b7 747b71c3
  6f8851c6 ccb45805 b6e2142f d7c81b4c 9ad22788 d2d4217c 61324451 1a93c1e6
  c351738d 10730ed3 7bb565ed 00e549fa c6d21ed7 99c0a3af a497060c 0795ee49
```

Similarly, for the second accumulator the key is as follows:

$p'' =$

```
0xbd62ffd5 78328d1b 415b2fda 68bdbfd0 acab107a f4970a65 7958808a 4a933b82
  a7deb9f9 f15c8bd2 2c9252dd a83b052b 142aaaf6 c8a09952 082f228a 421d7bcb
  4e7083ca 2158b88d c47fa50a b6ff3a8f a37a6060 4a30f5de 67fbc536 a3fa4fef
  3d81a08a 0c840375 5e241a8e 624d137c 5e868618 911d7d82 ba24653a 76b697d7
```

$q'' =$

```
0xb733e1b7 5ef5199e 427ea869 c0d88f93 f2ea947c 0ca47479 30b37e8c 562cfad9
  3a46c0c8 9d6ef679 e0a48169 dff2c35a 14a44339 b116e837 b532e82f c4c2c5b7
  63255324 d21055ad d703b6c2 96cc22d7 ff831008 b1e23182 faeb711c 1cf8ee63
  d3685621 402cb5c7 bee6bb8c 6e1b8bf0 6ec0acbc 03fcf407 f38947ee f460d937
```

$sk''_{ACC} =$

```
0x87882696 3b5241d0 eb039740 dae82177 edfe0684 3e2fe005 577d10d2 b7afbd18
  6dd64907 ee324826 eabeb8bd 4e18c5eb 819535a6 3d756b40 606ce16f 68d684d4
  7583fa97 ec4392ef 4f4ddb46 80db6fff 83cf36c0 33b8b4d6 4d166701 4163c159
  07c5d769 28e14af6 a3df0382 136d8608 f542a309 4e8e8433 3ba96b96 cc84a9ee
  5d4b5552 146749e1 0c8395f2 0e009bbd 69304e8f 565a170a 0efa1a4d d445f814
  1a613554 cbc4255a ad40613b dee801b8 febcd30f 9612c0a9 211a932f 7d465fcd
  10dda64a 8b2b114c 88049b1e 3165b5b0 3236177c ed5d1206 fd1f1adf 108829a6
  2a8970ca 7cf48e9a c43c0318 f0bfad21 d1788406 574d1376 31f1f91a b9786d24
```

$pk''_{ACC} =$

```
0x87882696 3b5241d0 eb039740 dae82177 edfe0684 3e2fe005 577d10d2 b7afbd18
  6dd64907 ee324826 eabeb8bd 4e18c5eb 819535a6 3d756b40 606ce16f 68d684d4
  7583fa97 ec4392ef 4f4ddb46 80db6fff 83cf36c0 33b8b4d6 4d166701 4163c159
  07c5d769 28e14af6 a3df0382 136d8608 f542a309 4e8e8433 3ba96b96 cc84a9ef
  d1e236de eb8ef09a 905d6e36 3796eb22 08c5f386 579595e8 b9061964 75062e6f
  fc86b017 5a8fa7a6 ba773583 6715ca3e 278bc140 0fca4232 de7c9de9 8426a14f
  c2737d39 7e941f88 2387f6eb 7f311317 d53387e5 e9703968 60065131 d17b67f9
  3b736775 c9a547d7 e146d933 c1284c8e 9ebfb6da ec678500 df9fa644 248fde31
```

According to the key generation process in the scheme specification, this yields:

$$ak=\left(sk_{DSS},sk'_{ACC},sk''_{ACC},pk'_{ACC},pk''_{ACC}\right)$$
$$rk=vk=\left(pk_{DSS},pk'_{ACC},pk''_{ACC}\right)$$

### D.4.3 Redactable attestation process

The message to be attested is defined as follows:

*m* = (“This is a test for ”, “the DPSS redactable attestation scheme as defined in ”, “ISO/IEC 23264-2.”)

The message blocks are thus defined as follows:

$m_1 =$ “This is a test for ”

$m_2 =$ “the DPSS redactable attestation scheme as defined in ”

$m_3 =$ “ISO/IEC 23264-2.”

NOTE The last character of $m_1$ and $m_2$ is a whitespace.

Accordingly, it holds that *n*=3. For this numerical example, let *adm*={1,3}, i.e. the first and last message blocks cannot be redacted.

Following the process description, the following values are obtained for *i*=1:

$r_1 =$

```
0xd27d0963 a26f2cc0 bdb7c2c1 2cfb8ad0 45f10279 65ba01cd 2f56000b a983f072
```

Setting $\left(acc'_1,aux'_1\right)=AEval\left(\left(sk'_{acc},pk'_{acc}\right),\{r_j\}_{j=1}^{1}\right)$ the following values are obtained

$acc'_1 =$

```
0x4c4493c8 af4bd737 2ee5aebf 18014142 6e7a0fd1 7a8f804f f700f672 3a56dc9a
  5b93220a f1eba1f0 884b288a f79203dc 15d5d028 f7a7ab36 453206da 132ba507
  5c17c635 d003d164 9a650dd2 7d5a1f7b f35646e8 3c055091 9d8fc9af f09db02e
  85dc9513 1311d1c6 9a7488b3 b2198d60 c76495c7 36b27597 25acdc65 6b89d8da
  a331912c 5e3f3d24 974c9a74 ac47084e a8ecc206 438b5a85 efe79908 b1ba0c85
  af4830a2 59c658bf db464e7e 565ec789 53b18f7f 26a9ef4d 81e91d6e 4e58a218
  a8fd68b4 85807108 f28af24a 0cc72b80 a4412fec 08b7460c bab65ffb d1b49e38
  0d4c54c8 8130ff3f 8dfd0233 15811455 fa1cc16d cb599f10 8f90ce28 21be0755
```

$aux'_1 =$ Null (empty string)

For *i*=1, the inner loop only has *j*=1:

$wit'_{11}=AWitCreate\left(\left(sk'_{acc},pk'_{acc}\right),acc'_1,aux',r_1\right)=$

```
0x6fa31d57 3218752c f3180184 c3f1f4fd e32f445b 9a67091e 494bd187 fbfa29c3
  39f5b3f7 7d7f34b0 e6adbaf5 3b7a0639 24b8e35e a99c55c3 c22e16df 3666d27e
  a1b4f2bc db9e634e e78c0c67 90b4d7ed ce42a642 630736a3 d702dc7d b1820cde
  1f1e9eff 4d1e4009 90b50edd d0d3333e c91b1d7d ba71d03c 9b9ed297 cd3de265
  f668491c fc8e3260 e2e96cdd b02122dc 97f81463 638b0b7d fa990b4b 0ae8782c
  9390fbdf 82b59647 866808a4 d36d6b89 7cbb4be1 eaaa9091 edb78c83 811b8a28
  deaae3b6 0e741384 1da4016e e62316f6 0fd25cf8 fcf4a15b 6e9259e9 bd1c791a
  a9553a47 b11645ee d234a7a5 074ee80d 41a7e787 964349ea 8e112ff1 f4deed55
```

Finally, the algorithm sets:
$WIT'_1=\left(wit'_{11}\right)$

For *i*=2, the following values are computed: