# Redactable Signature Schemes for Trees With Signer-Controlled Non-Leaf-Redactions*

Hermann de Meer[1,3], Henrich C. Pöhls[2,3]**,
Joachim Posegga[2,3] Kai Samelin***

[1] Chair of Computer Networks and Communications, University of Passau, Germany
[2] Chair of IT-Security, University of Passau, Germany
[3] Institute of IT-Security and Security Law (ISL), University of Passau, Germany
demeer@fim.uni-passau.de, {hp,jp,ks}@sec.uni-passau.de

**Abstract.** Redactable signature schemes (RSS) permit to remove parts from signed documents, while the signature remains valid. Some RSSs for trees allow to redact non-leaves. Then, new edges have to be added to the tree to preserve it's structure. This alters the position of the nodes' children and may alter the semantic meaning encoded into the tree's structure. We propose an extended security model, where the signer explicitly controls among which nodes new edges can be added. We present a provably secure construction based on accumulators with the enhanced notions of indistinguishability and strong one-wayness.

## 1 Introduction

Trees are commonly used to structure data. XML is one of today's most prominent examples. To protect these documents against unauthorized modifications, digital signatures are used. They protect two important properties: integrity of the data itself and also the data's origin. In certain scenarios it is desirable to *remove* parts of a signed document without invalidating the protecting signature. However, classical signatures prevent any alteration of data.

The straight-forward solution to this problem is to request a new signature with the parts in question removed. This round-trip allows to satisfy the above requirements. However, what happens if the original signer is not reachable, or communication is too costly? The "digital document sanitization problem" [35] therefore asks for two additional requirements: (1) the original signer must not be involved for derivation of signatures, and (2) the removed parts must remain private. This is also useful in cases where the signer must not know which parts

* This is an extended and heavily revised version of [40]

** The research leading to these results has received support from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 609094.

*** Was supported by "Regionale Wettbewerbsfähigkeit und Beschäftigung", Bayern, 2007-2013 (EFRE) as part of the SECBIT project (http://www.secbit.de) and the European Community's Seventh Framework Programme through the EINS Network of Excellence under grant agreement n° 288021, while at the University of Passau.

of a signed document are passed to other parties. Redactable signature schemes (RSS) address the above constellation. As standard signatures schemes, they prohibit *unauthorized* changes: only removal is allowed. This possibility comes in handy in many scenarios, e.g., privacy-preserving handling of medical records becomes simpler [32, 41, 46, 48]. There are many more applications given in the literature. [7, 27, 42] provide additional scenarios.

**State of the Art and Related Work.** The concept of RSSs has been introduced in [31, 47]. Both describe a signature scheme that allows removing parts from signed data without invalidating the signature. [31] termed this functionality "redactable signatures". Constructions emerged in the following years calling this same functionality differently, e.g., in the following work it is termed close to or as "sanitizing" [29, 30, 35–37]. For example, *Izu* et al. call this functionality "sanitized signatures" [29, 30]. We follow the terminology from [31], describing the functionality as "redactable signatures" [31]. Hence, this paper disagrees with the classification from [30], which states that [47] are "sanitized signatures". This is especially important for a clear separation from the concept of "sanitizable signature schemes" [2, 11, 13, 14, 24, 33], coined by *Ateniese* et al. [2]. They are, however, to some extend, related. In sanitizable signatures, elements are not redacted, but (admissible) ones can be altered to arbitrary strings. To do so, sanitizers require to know a secret. Even though the primitives seem to be very related, the aims and security models substantially differ on a detailed level [21, 39].

Following the first ideas, RSSs have been proposed to work for lists [19, 44], and have extended to trees [12, 32] and graphs [32]. *Brzuska* et al. derived a set of desired properties for redactable tree-structured documents including a formal model for security notions [12]. Following their definitions, most of the schemes proposed are not secure, e.g., the work done in [26, 31, 32, 35, 47, 48]. In particular, a third party can see that something has been redacted, which impacts on the intention of an RSS. However, their model is limited to leaf-redaction only.

Recently, schemes with *context-hiding*, a very strong privacy notion, and variations thereof, e.g., [1, 3, 4] appeared. In those schemes, a derived signature does not leak whether it corresponds to an already existing signature in a statistical sense. Most recent advances generalize similar ideas, e.g., [1, 3–5, 9, 10].

**Flexibility of Non-Leaf Redactions.** Consider the tree depicted in Fig. 1, ignoring the numbers in brackets for now. To remove the leaf $n_4$, the node $n_4$ itself and the edge $e_{3,4}$ is removed. By consecutive removal of leaves, complete subtrees can be redacted [12]. However, schemes only allowing redaction of leaves fail to redact the data stored in, e.g., $n_3$ only. The wanted tree is depicted in Fig. 3: to connect $n_4$ to the remaining tree, the third party requires to add a *new* edge $e_{1,4}$, which was not present before. However, $e_{1,4}$ is in the *transitive closure* of the original tree, as shown in Fig. 2. The scheme introduced in [32] allows redaction of non-leaves, stating that this flexibility is useful in many scenarios.

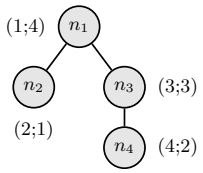


**Fig. 1.** Original Tree with Traversal Numbers

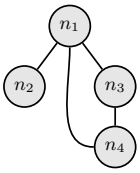


**Fig. 2.** Transitive closure of the child-of relation

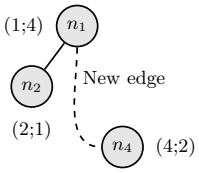


**Fig. 3.** After removal of $n_3$; with orig. traversal numbers

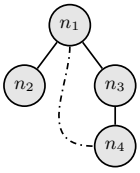


**Fig. 4.** Added explicitly authorized potential edge

One may think of redacting hierarchies. We model non-leaf redaction as a two step process: first, all children of the to-be-redacted node are re-located to its parent. The to-be-redacted node is now a leaf and can be redacted as such. Allowing non-leaf removal has its merits, but generally allowing this behavior can lead to a reduced structural integrity protection, as we describe next.

**Structural Integrity Protection.** Let us consider a chart encoding employees' names as nodes and their position within the companies hierarchy is encoded in the trees structure. Hence, protecting structural integrity is equal to protecting the correctness of the employees' hierarchical positions. If one only signs the ancestor relationship of the nodes, all edges that are part of the transitive closure are part of the signature. This is depicted in Fig. 2. This allows a third party to add edges to the tree. This possibility was named "Level Promotion" in [44]. In our prior example, this translates easily: an employee can be "promoted". This may not *always* be wanted.

The scheme introduced in [32] behaves like this: it builds upon the idea that having all pre- and post-order traversal numbers of the nodes in a tree, one can uniquely reconstruct it. To make their scheme hiding occurred redactions, the traversal numbers are randomized in an order-preserving manner, which does not have an impact on the reconstruction algorithm, as the relation between nodes does not change. For our discussion, this step can be left out.[4] Assume we redact $n_3$, as depicted in Fig. 3: the traversal-numbers are still in the correct relation. Hence, the edge $e_{1,4}$, which has not explicitly been present before, passes verification. One might argue that nesting of elements must adhere to a specific codified structure, i.e., XML-Schema. Henceforth, possibilities like level-promotions are detected by any XML-Schema validation. However, elements may contain itself, like hierarchically structured employees or treatments composed of treatments. Hence, redaction of non-leaves is not acceptable in the generic case and may lead to several new attack vectors, similar to the ones of XPath [25]. We conclude that the signer must *explicitly* sign only the authorized transitive edges, if the aforementioned behavior is not wanted, or use an RSS which only permits leaf-redactions.

[4] Indeed, the randomization step does not hide anything [12, 43].

**Our Contribution.** We present a security model where the signer has the flexibility to allow redaction of any node. Our model allows level promotions due to re-locations of specified sub-trees, which resembles the *implicit* possibility of previous schemes. The signer is *explicitly* prohibiting the redaction of nodes individually, as the signer must explicitly sign an edge for re-locations. Re-locations of sub-trees can be used to emulate non-leaf redactions, but allow even more flexibility: we can relocate sub-trees without redactions. We also allow that a sanitizer can prohibit such re-locations by redacting the authorized potential edge.

While [43] either allows or disallows non-leaf redactions completely, this work allows the signer to decide which non-leaves can be redacted: the signer defines to which "upper-level node" the "dangling" sub-tree's root can be connected to.

We derive a provably secure construction, based on cryptographic accumulators [6, 8], in combination with *Merkle*'s Hash-Tree-Technique. Thus, our construction requires only standard cryptographic primitives. However, we need to strengthen existing definitions of accumulators. In particular, we introduce the notions of indistinguishability and strong one-wayness of accumulators.

In our construction, the signer controls the protection of the order of siblings. Hence, our scheme is capable of signing both ordered and unordered trees. Finally, we present some new attacks on existing schemes.

## 2 Preliminaries and Security Model

Nodes are addressed as $n_i$. The root is denoted as $n_1$. With $c_i$, we refer to all the content of node $n_i$, which is additional information that might be associated with a node, i.e., data, element name and so forth. We use the work done in [12] as our starting point. Their model only allows removing *a single leaf* at a time and does not support non-leaf redactions.

**Flexible RSS.** An RSS consists of four efficient (PPT) algorithms: $\mathcal{RSS} := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Modify})$. All algorithms output $\perp$ in case of an error. Also, they take an implicit security parameter $\lambda$ (in unary).

KeyGen. The algorithm KeyGen outputs the key pair of the signer, i.e., $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\lambda$ being the security parameter.

Sign. On input of $\mathtt{sk}$, $T$, and $\mathtt{ADM}$, Sign outputs a signature $\sigma$. $\mathtt{ADM}$ controls what changes by Modify are admissible. In detail, $\mathtt{ADM}$ is the set containing all signed edges, including the ones where a sub-tree can be re-located to. In particular, $(n_i, n_j) \in \mathtt{ADM}$, if the edge $(n_i, n_j)$ must verify. These edges cannot be derived from $T$ alone. Let $(T, \sigma, \mathtt{ADM}) \leftarrow \mathsf{Sign}(\mathtt{sk}, T, \mathtt{ADM})$.

Verify. On input of $\mathtt{pk}$, the tree $T$ and a signature $\sigma$, Verify outputs a bit $d \in \{0, 1\}$, indicating the validity of $\sigma$, w.r.t. $\mathtt{pk}$ and $T$: $d \leftarrow \mathsf{Verify}(\mathtt{pk}, T, \sigma)$. Note, $\mathtt{ADM}$ is not required.

Modify. The algorithm Modify takes $\mathtt{pk}$, the tree $T$, a signature $\sigma$ and $\mathtt{ADM}$, and an instruction $\mathtt{MOD}$. $\mathtt{MOD}$ contains the actual change to be made: redact a sub-tree, relocate a sub-tree, or prohibit relocating a sub-tree. On modification,

**Experiment** $\mathsf{Unforgeability}^{\mathsf{RSS}}_{\mathcal{A}}(\lambda)$
$(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(T^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathtt{sk},\cdot,\cdot)}(\mathtt{pk})$
let $i = 1, 2, \ldots, q$ index the queries/answers to/from $\mathsf{Sign}$
return 1, if
$\mathsf{Verify}(\mathtt{pk}, T^*, \sigma^*) = 1$ and
for all $1 \leq i \leq q$, $T^* \notin \mathrm{span}_{\vdash}(T_i, \sigma_i, \mathtt{ADM}_i)$

**Fig. 5.** Unforgeability

ADM is adjusted. If a node $n_i$ is redacted, the edge to its father needs to be removed. Moreover, if there exists a sub-tree which can be re-located under the redacted node, the corresponding edges need to be removed from ADM as well. The alteration of ADM is crucial to maintain privacy and transparency. Hence, we have: $(T', \sigma', \mathtt{ADM}') \leftarrow \mathsf{Modify}(\mathtt{pk}, T, \sigma, \mathtt{ADM}, \mathtt{MOD})$.

We require the usual correctness requirements to hold [12]. A word of clarification: we assume that ADM is always correctly derivable from $\sigma$. However, we always explicitly denote ADM to increase readability of our security definitions.

**The Extended Security Model.** We build around the framework given in [12], extending it to cater for the flexibility of non-leaf redactions and re-locations.

*Unforgeability:* No one should be able to compute a valid signature on a tree $T^*$ verifying for pk outside $\mathrm{span}_{\vdash}(T, \sigma, \mathtt{ADM})$, without access to the corresponding secret key sk. Here, $\mathrm{span}_{\vdash}(T_i, \sigma_i, \mathtt{ADM}_i)$ expresses the set of trees derivable by use of Modify on $T_i$, $\sigma_i$ and $\mathtt{ADM}_i$. This is analogous to the standard unforgeability requirement for signature schemes [23]. A scheme RSS is unforgeable, if for any PPT adversary $\mathcal{A}$, the probability that the game depicted in Fig. 5 returns 1, is negligible.

*Privacy:* No one should be able to gain any knowledge about parts redacted. This is similar to the standard indistinguishability notation for encryption schemes [22]. An RSS is private, if for any PPT adversary $\mathcal{A}$, the probability that the game shown in Fig. 6 returns 1, is negligibly close to $\frac{1}{2}$. In a nutshell, privacy says that everything which has been redacted remains hidden. However, if in real documents redactions are obvious, e.g., due to missing structure, one may trivially be able to decide that not the complete tree was given to the verifier. However, this cannot be avoided: our definitions assume that no other sources of knowledge apart from (several) $\sigma'_i$, $T'_i$ and $\mathtt{ADM}'_i$ are available to the attacker.

*Transparency:* A party who receives a signed tree $T$ should not be able to tell whether it received a freshly signed tree (case $b = 1$ in Fig. 7) or a tree derived by Modify [12]. We say that an RSS is transparent, if for any PPT adversary $\mathcal{A}$, the probability that the game shown in Fig. 7 returns 1, is negligibly close to $\frac{1}{2}$.

**Experiment** $\mathsf{Privacy}^{\mathsf{RSS}}_{\mathcal{A}}(\lambda)$
$(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$d \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathtt{sk},\cdot,\cdot),\mathsf{LoRModify}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\mathtt{sk},b)}(\mathtt{pk})$
where oracle $\mathsf{LoRModify}(T_{j,0}, \mathtt{ADM}_{j,0}, \mathtt{MOD}_{j,0}, T_{j,1}, \mathtt{ADM}_{j,1}, \mathtt{MOD}_{j,1}, \mathtt{sk}, b)$
if $\mathtt{MOD}_{j,0}(T_{j,0}) \neq \mathtt{MOD}_{j,1}(T_{j,1})$ return $\perp$
$(T_{j,0}, \sigma_0, \mathtt{ADM}_{j,0}) \leftarrow \mathsf{Sign}(\mathtt{sk}, T_{j,0}, \mathtt{ADM}_{j,0})$
$(T_{j,1}, \sigma_1, \mathtt{ADM}_{j,1}) \leftarrow \mathsf{Sign}(\mathtt{sk}, T_{j,1}, \mathtt{ADM}_{j,1})$
$(T'_{j,0}, \sigma'_0, \mathtt{ADM}'_{j,0}) \leftarrow \mathsf{Modify}(\mathtt{pk}, T_{j,0}, \sigma_0, \mathtt{ADM}_{j,0}, \mathtt{MOD}_{j,0})$
$(T'_{j,1}, \sigma'_1, \mathtt{ADM}'_{j,1}) \leftarrow \mathsf{Modify}(\mathtt{pk}, T_{j,1}, \sigma_1, \mathtt{ADM}_{j,1}, \mathtt{MOD}_{j,1})$
if $\mathtt{ADM}'_{j,0} \neq \mathtt{ADM}'_{j,1}$, abort returning $\perp$
return $(T'_{j,b}, \sigma'_b, \mathtt{ADM}'_{j,b})$
return 1, if $b = d$

**Fig. 6.** Privacy

**Experiment** $\mathsf{Transparency}^{\mathsf{RSS}}_{\mathcal{A}}(\lambda)$
$(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$d \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathtt{sk},\cdot,\cdot),\mathsf{ModifyOrSign}(\cdot,\cdot,\cdot,\mathtt{sk},b)}(\mathtt{pk})$
where oracle $\mathsf{ModifyOrSign}(T, \mathtt{ADM}, \mathtt{MOD}, \mathtt{sk}, b)$
if $\mathtt{MOD} \notin \mathtt{ADM}$, return $\perp$
$(T, \sigma, \mathtt{ADM}) \leftarrow \mathsf{Sign}(\mathtt{sk}, T, \mathtt{ADM})$
$(T', \sigma', \mathtt{ADM}') \leftarrow \mathsf{Modify}(\mathtt{pk}, T, \sigma, \mathtt{ADM}, \mathtt{MOD})$
if $b = 1$:
$(T', \sigma', \mathtt{ADM}') \leftarrow \mathsf{Sign}(\mathtt{sk}, T', \mathtt{ADM}')$
return $(T', \sigma', \mathtt{ADM}')$
return 1, if $b = d$

**Fig. 7.** Transparency

*Relations:* The implications and separations between the security properties given in [12] do not change — the proofs are very similar and therefore omitted in this work. In particular, transparency implies privacy, while transparency and unforgeability are independent.

**Cryptographic Accumulators.** For our construction, we deploy accumulators. They have been introduced in [8]. The basic idea is to hash a set $\mathcal{S}$ into a short value $a$, normally referred to as the accumulator. For each element $y_i \in \mathcal{S}$ a short witness $w_i$ is generated, which allows to verify that $y_i$ has actually been accumulated into $a$. We only need the basic operations of an accumulator, e.g., neither trapdoor-freeness [34, 45] nor dynamic updates [18], or revocation tech-

**Experiment** $\mathsf{Strong-One-Wayness}_{\mathcal{A}}^{\mathcal{AH}}(\lambda)$
$\mathtt{pk} \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(a^*, y^*, p^*) \leftarrow \mathcal{A}^{\mathsf{Hash}(\mathtt{pk},\cdot)}(1^\lambda, \mathtt{pk})$
where oracle $\mathsf{Hash}$ for input $\mathcal{S}_i$:
$(a_i, \mathsf{aux}_i) \leftarrow \mathsf{Hash}(\mathtt{pk}, \mathcal{S}_i)$
return $(a_i, \{(y_j, p_j) \mid y_j \in \mathcal{S}_i, p_j \leftarrow \mathsf{Proof}(\mathtt{pk}, \mathsf{aux}_i, a_i, y_j, \mathcal{S}_i)\})$
look for $k$ s.t. $a_k = a^*$. If such $k$ does not exist, return 0.
return 1, if $\mathsf{Check}(1^\lambda, \mathtt{pk}, y^*, p^*, a^*)$ and $y^* \notin \mathcal{S}_k$

**Fig. 8.** Accumulator Strong One-Wayness

niques [17] are required. A basic accumulator consists of four efficient algorithms, i.e., $\mathcal{AH} := \{\mathsf{KeyGen}, \mathsf{Hash}, \mathsf{Proof}, \mathsf{Check}\}$:

$\mathsf{KeyGen}$**.** Outputs the public key $\mathtt{pk}$ on input of a security parameter $\lambda$:
$\mathtt{pk} \leftarrow \mathsf{KeyGen}(1^\lambda)$

$\mathsf{Hash}$**.** Outputs the accumulator $a$, and an auxiliary value $\mathsf{aux}$, given a set $\mathcal{S}$, and $\mathtt{pk}$: $(a, \mathsf{aux}) \leftarrow \mathsf{Hash}(\mathtt{pk}, \mathcal{S})$

$\mathsf{Proof}$**.** On input of an auxiliary value $\mathsf{aux}$, the accumulator $a$, a set $\mathcal{S}$, and an element $y \in \mathcal{S}$, $\mathsf{Proof}$ outputs a witness $w$, if $y$ was actually accumulated:
$w \leftarrow \mathsf{Proof}(\mathtt{pk}, \mathsf{aux}, a, y, \mathcal{S})$

$\mathsf{Check}$**.** Outputs a bit $d \in \{0, 1\}$, indicating if a given value $y$ was accumulated into the accumulator $a$ with respect to $\mathtt{pk}$ and a witness $w$:
$d \leftarrow \mathsf{Check}(\mathtt{pk}, y, w, a)$

All correctness properties must hold [6]. Next, we define the required security properties of accumulators.

*Strong One-Wayness of Accumulators.* It must be hard to find an element not accumulated, even if the adversary can chose the set to be accumulated. The needed property is strong one-wayness of the accumulator [6]. We say that an accumulator is strongly one-way, if the probability that the game depicted in Fig. 2 returns 1, is negligibly close to 0. Note, in comparison to [6, 38], we consider probabilistic accumulation and allow to query adaptively.

*Indistinguishability of Accumulators.* We require that an adversary cannot decide how many additional members have been digested. We say that an accumulator is indistinguishable, if the probability that the game depicted in Fig. 2 returns 1, is negligibly close to $\frac{1}{2}$. Here, the adversary can choose three sets, and has to decide, which sets have been accumulated (either the first and the second, or the first and the third). Note, only the witnesses for the first set are returned. An accumulator not fulfilling these requirements has been proposed by *Nyberg* in [38]; the underlying *Bloom*-Filter can be attacked by probabilistic methods and therefore leaks the amount of members [20]. This is not acceptable for our construction, as it impacts on privacy. A concrete instantiation of such

**Experiment** $\mathsf{Indistinguishability}_{\mathcal{A}}^{\mathcal{AH}}(\lambda)$
$\mathtt{pk} \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$d \leftarrow \mathcal{A}^{\mathsf{LoRHash}(\cdot,\cdot,\cdot,b,\mathtt{pk})}(1^\lambda, \mathtt{pk})$
where oracle $\mathsf{LoRHash}$ for input $\mathcal{S}, \mathcal{R}_0, \mathcal{R}_1$:
$(a, \mathsf{aux}) \leftarrow \mathsf{Hash}(\mathtt{pk}, \mathcal{S} \cup \mathcal{R}_b)$
return $(a, \{(y_i, p_i) \mid y_i \in \mathcal{S}, p_i \leftarrow \mathsf{Proof}(\mathtt{pk}, y_i, \mathsf{aux})\})$
return 1, if $d = b$

**Fig. 9.** Accumulator Privacy

an accumulator achieving our requirements is the probabilistic version of [6]. In a nutshell, instead of fixing the base for the RSA-function, it is drawn at random. A more detailed discussion is given in [20]. We do note that our definition of indistinguishability already assumes a probabilistic hash algorithm; [20] also accounts for deterministic ones. Additional information about accumulators can be found in [6, 8, 18].

## 3 A new Flexible RSS

Our construction makes use of *Merkle*-Hash-Trees. The *Merkle*-Hash $\mathcal{MH}$ of a node $x$ is calculated as: $\mathcal{MH}(x) = \mathcal{H}(\mathcal{H}(c_x)||\mathcal{MH}(x_1)|| \ldots ||\mathcal{MH}(x_n))$, where $\mathcal{H}$ is a collision-resistant hash-function, $c_x$ the content of the node $x$, $x_i$ a child of $x$, $n$ the number of children of the node $x$, while $||$ denotes a uniquely reversible concatenation of strings. $\mathcal{MH}(n_1)$'s output depends on all nodes' content and on the *right order* of the siblings. Hence, signing $\mathcal{MH}(n_1)$ protects the integrity of the nodes in an ordered tree and the tree's structural integrity. Obviously, this technique does not allow to hash unordered trees: an altered order most likely causes a different digest value.

*Hash-Trees and Privacy.* Removing sub-trees requires to give a hash of the removed node to the verifier, in order to calculate the same $\mathcal{MH}(n_1)$. This directly impacts on privacy and transparency, because the hash depends on removed information that shall remain private. One example for an RSS which suffers from this problem is given in [28]. It can be attacked in the following way: the attacker asks its left-or-right oracle to sign a root with one child only, but without redacting anything. The other input is a tree with the root and two children, while the *left* child is to redacted. This results in the same tree: the root with one child. However, in the case the first input is used, their "fake-digest" is the right node, while in the other case the fake-digest is the left node. This can clearly be distinguished and privacy is broken.

A more detailed analysis of the *Merkle*-Hash-Tree is given in [32], which also gives an introduction on the possible attacks on non-private schemes. To

overcome the limitation of *Merkle*-Hash-Trees, we use accumulators instead of standard collision-resistant hash-functions. We do note that the idea to use accumulators has already been proposed in [32]. However, they state that accumulators are not able to achieve the desired functionality. We show that they are sufficient by giving a concrete construction. Note, compared to the old version of this paper [40], we do not permit a redaction of the root, as this may lead to problematic behavior as well [43]. Moreover, this small constraint reduces the number of signatures to be generated to only one.

**Construction.** We allow explicit re-location of sub-trees. If a non-leaf is subject to redaction, all sub-trees of the node need to be re-located. If this is possible and what their new ancestor will be must be under the sole control of the signer. We limit re-locations directing towards the root to avoid forming loops, which was possible in the original publication [40]. We now sketch our solution, and give the concrete algorithms afterward. Our re-location definition does not require to delete the ancestor node. This behavior of re-locating only is discussed later on.

*Sketch.* In our solution, the signer replicates all re-locatable nodes and the underlying sub-trees to all locations where a sanitizer is allowed to relocate the sub-tree to. The replicas of the nodes are implicitly used to produce the re-locatable edges. Each additional edge is contained in $\texttt{ADM}$. To prohibit simple copy attacks, i.e., leaving a re-located sub-tree in two locations, each node $n_i$ gets an associated unique nonce $r_i$. The whole tree gets signed using a *Merkle*-Hash-Tree, but using an accumulator instead of a standard hash. To redact parts, the sanitizer removes the nodes in question, and no longer provides the corresponding witnesses. As accumulators work on sets, it does not matter in what order the members are checked. However, if ordered trees are present, the ordering between siblings has to be explicitly signed. To do so, we sign the "left-of" relation, as already used and proposed in [12, 19, 44]. Note, this implies a quadratic complexity in the number $n$ of siblings, i.e, $\frac{n(n-1)}{2}$. To relocate a sub-tree, one only applies the necessary changes to $T$, without any further changes. However, a sanitizer can prohibit consecutive re-locations by altering $\texttt{ADM}$. This control is similar to consecutive sanitization control [35]. Verification is straight forward: for each node $x$ inside the tree check, if $x$'s content, $x$'s children and $x$'s order to other siblings is contained in $x$'s *Merkle*-Hash. This is done recursively. Further, all node's nonces must be unique for this tree. Finally, the root's signature is checked.

**The Algorithmic Description.** $\Pi := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ denotes a standard unforgeable signature scheme [23]. Note, to shorten the *algorithmic* description, we abuse notation and define that $\mathsf{Hash}$ directly works on a set and returns all witness/element pairs $(w_i, y_i)$. We denote the accumulation as $(a, \mathcal{W} = \{(w_i, y_i)\}) \leftarrow \mathcal{AH}(\texttt{pk}, \{y_1, \ldots, y_n\})$. We use *//comment* to indicate comments.

**$\mathsf{KeyGen}(\lambda)$:**

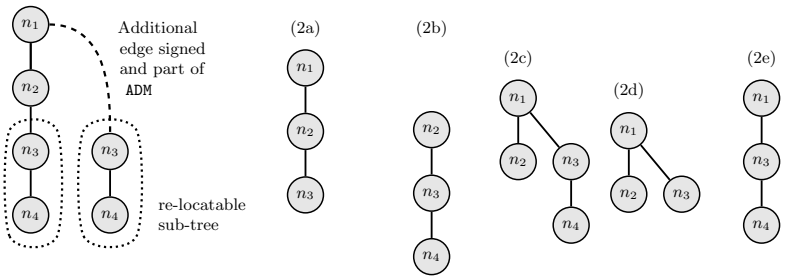


**Fig. 10.** Left: expanded tree with duplicates, (2a-e) Examples of valid trees after redactions or re-locations.

$\mathtt{pk}_{\mathcal{AH}} \leftarrow \mathcal{AH}.\mathsf{KeyGen}(1^\lambda)$
$(\mathtt{pk}_S, \mathtt{sk}_S) \leftarrow \Pi.\mathsf{KeyGen}(1^\lambda)$
return $((\mathtt{pk}_S, \mathtt{pk}_{\mathcal{AH}}), \mathtt{sk}_S)$

**Expand**$(T, \mathtt{ADM})$**:**
For all edges $e_i \in \mathtt{ADM} \setminus T$ (must be done *bottom-up*)
Replicate the sub-tree underneath the node addressed by $e_i$
to the designated position. *//Note: this is recursive!*
Return this expanded tree

**Sign**$(\mathtt{sk}, T, \mathtt{ADM})$**:**
*//We implicitly assume a parameter* $s \in \{\text{ordered}, \text{unordered}\}$,
*//denoting if the order must be protected*
For each node $n_i \in T$:
$r_i \xleftarrow{\$} \{0,1\}^\lambda$
Append $r_i$ to each node $n_i \in T$
Expand tree: $\Omega \leftarrow \mathsf{Expand}(T, \mathtt{ADM})$ *//Note: $r_i$ is copied as well*
Do the next step with the expended tree $\Omega$:
If $s =$ unordered: *//$\mathcal{MH}(\cdot)$ denotes the digest calculated by $\mathcal{AH}$*
$(d_1, \{(y_k, w_k)\}) \leftarrow \mathcal{AH}(\mathtt{pk}, \{c_1||r_1, \mathcal{MH}(x_1), \ldots, \mathcal{MH}(x_n)\})$
Else ($s =$ ordered): *//ordered tree*
$(d_1, \{(y_k, w_k)\}) \leftarrow \mathcal{AH}(\mathtt{pk}, \{c_1||r_1, \mathcal{MH}(x_1), \ldots, \mathcal{MH}(x_n), \Xi_x\})$,
where $\Xi_x = \{r_i||r_j \mid 0 < i < j \leq n\}$
Sign the root-hash: $\sigma_s \leftarrow \Pi.\mathsf{Sign}(\mathtt{sk}_S, d_1||s)$
$\mathcal{W} = \{(y_k, w_k)\}$ denotes the set of **all** witness/element pairs returned
return $\sigma = (\sigma_s, \mathcal{W}, \mathtt{ADM})$

**Modify**$(\mathtt{pk}, T, \sigma, \mathtt{ADM}, \mathtt{MOD})$**:**
use Verify to verify the tree $T$
Expanded tree $\Omega \leftarrow \mathsf{Expand}(\mathsf{T}, \mathtt{ADM})$
Case 1: MOD instruction to redact sub-tree $T_s$ (only via leaf-redaction):
*//1. remove all $n_l \in T_s$ (incl. replicas) from $\Omega$:*
Set $\Omega' \leftarrow \Omega \setminus n_l$

*//2. remove all $n_l \in T_s$ from $T$:*
Set $T' \leftarrow T \setminus T_s$
Create $\texttt{ADM}'$ by removing all ingoing edges all nodes in $T_s$ from $\texttt{ADM}$
return $\sigma' = (T', \sigma_s, \mathcal{W} \setminus \{(y_k, w_k) \mid y_k \in \Omega'\}, \texttt{ADM}')$

Case 2: $\texttt{MOD}$ instruction to re-locate $T_s$:
Set $T' \leftarrow \texttt{MOD}(T)$
return $\sigma$

Case 3: $\texttt{MOD}$ instruction to remove re-location edges $e$:
Set $\texttt{ADM}' \leftarrow \texttt{ADM} \setminus e$
*//Note: This expansion is done with the modified $\texttt{ADM}'$.*
Let $\Omega' \leftarrow \mathsf{Expand}(T, \texttt{ADM}')$
return $\sigma' = (T, \sigma_s, \mathcal{W} \cap \{(y_k, w_k) \mid y_k \in \Omega'\}, \texttt{ADM}')$

**$\mathsf{Verify}(\texttt{pk}, T, \sigma)$:**
Check if each $r_i \in T$ is unique.
Check $\sigma$ using $\Pi.\mathsf{Verify}$
Let the value protected by $\sigma_s$ be $d'_1 = d_1 || s$
For each node $x \in T$:
For all children $x_i$ of $x$ do:
*//Note: checks if children are signed*
Let $d \leftarrow \mathsf{Check}(\texttt{pk}, d_i, w_i, d_x)$ *//$d_x$ denotes the node's digest*
If $d = 0$, return 0
If $s =$ ordered:
*//Is every "left-of"-relation signed?*
*//Note: only linearly many checks*
For all $0 < i < n$:
$d \leftarrow \mathsf{Check}(\texttt{pk}, r_i || r_{i+1}, w_{x,x+1}, d_x)$
If $d = 0$, return 0
return 1

Arguably, allowing re-location without redaction may also be too much freedom. However, it allows the signer to allow a flattening of hierarchies, i.e., to remove the hierarchical ordering of treatments in a patient's record. We want to stress that copying complete sub-trees may lead to an exponential blow-up in the number of nodes to the signed. This happens, in particular, if re-locations are nested. However, if only used sparely, our construction remains useable, as a performance analysis shows next.

**Performance.** We have implemented our scheme to demonstrate its usability using the old algorithm given in [40], i.e., where every accumulator is signed, not only the root. As the accumulator, we chose the original construction [8] in its randomized form. Tests were performed on a *Lenovo Thinkpad T61* with an *Intel* T8300 Dual Core @`2.40` GHz and `4` GiB of RAM. The OS was *Ubuntu* Version 10.04 LTS (64 Bit) with Java-Framework `1.6.0_26-b03` (OpenJDK). We took the median of 10 runs: we only want to demonstrate that our construction is practical as a proof-of-concept. We measured trees with unordered siblings and

| | Generation of $\sigma$ | | | Verification of $\sigma$ | | |
|---|---|---|---|---|---|---|
| Nodes | 10 | 100 | 1,000 | 10 | 100 | 1,000 |
| Ordered | 276 | 6,715 | 57,691 | 26 | 251 | 2,572 |
| Unordered | 103 | 599 | 5,527 | 21 | 188 | 1,820 |
| SHA-512 | 4 | 13 | 40 | 4 | 13 | 40 |

**Table 1.** Median Runtime in ms

one with ordered siblings. Trees were randomly generated in an iterative fashion. Re-locations were not considered: only leaf-removal has been implemented. Time for generation of keys for the hash is included. We excluded the time for creating the required signature key pair. However, both becomes negligible in terms of the performance for large trees. On digest calculation, we store all intermediate results in RAM to avoid any disk access impact.

As shown, our construction runtime remains within useable limits. The advanced features come at a price; our scheme is considerably slower than a standard hash like SHA-512. Signatures are more often verified than generated, so the overhead for verification has a greater impact. All other provable secure and transparent schemes, i.e., [12] and [19], have the same complexity and therefore just differ by a constant factor. [12] and [19] do not provide a performance analysis on real data. Compared to [43], where a performance analysis of a prototype is provided, this construction offers equal speed or is faster.

**Security of the Construction.** Our scheme is unforgeable, private and transparent. Assuming $\mathcal{AH}$ is strongly one-way, and the signature scheme $\Pi$ is UNF-CMA, our scheme is unforgeable, while the indistinguishability of $\mathcal{AH}$ implies privacy and transparency. The formal proofs are relegated to App. A.

## 4 Conclusion

We have shown that redacting arbitrary nodes of a tree can lead to severe problems. Our security model captures that the signer has to explicitly mark redactable nodes. We derived a new construction based on accumulators. Our construction can handle ordered and unordered trees. We have implemented our scheme, and as our performance measurements show, it is reasonably fast. It remains unclear how we can make RSSs accountable [15, 16], and if more efficient schemes exist.

## References


1. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. Computing on authenticated data. ePrint Report 2011/096, 2011.
2. G. Ateniese, D. H. Chou, B. De Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS*, pages 159–177. Springer, 2005.

3. N. Attrapadung, B. Libert, and T. Peters. Computing on authenticated data: New privacy definitions and constructions. In *ASIACRYPT*, pages 367–385, 2012.
4. N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *PKC*, pages 386–404, 2013.
5. M. Backes, S. Meiser, and D. Schröder. Delegatable functional signatures. *IACR Cryptology ePrint Archive*, 2013:408, 2013.
6. N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, pages 480–494, 1997.
7. A. Becker and M. Jensen. Secure combination of xml signature application with message aggregation in multicast settings. In *ICWS*, pages 531–538, 2013.
8. J. C. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures (extended abstract). In *EUROCRYPT*, pages 274–285, 1993.
9. D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149–168, 2011.
10. E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. *IACR Cryptology ePrint Archive*, 2013:401, 2013.
11. C. Brzuska et al. Security of Sanitizable Signatures Revisited. In *Proc. of PKC 2009*, pages 317–336. Springer, 2009.
12. C. Brzuska et al. Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In *ACNS*, pages 87–104. Springer, 2010.
13. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Sanitizable signatures: How to partially delegate control for authenticated data. In *Proc. of BIOSIG*, volume 155 of *LNI*, pages 117–128. GI, 2009.
14. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of sanitizable signatures. In *Public Key Cryptography*, pages 444–461, 2010.
15. C. Brzuska, H. C. Pöhls, and K. Samelin. Non-Interactive Public Accountability for Sanitizable Signatures. In *EuroPKI*, volume 7868 of *LNCS*, pages 178–193, 2012.
16. C. Brzuska, H. C. Pöhls, and K. Samelin. Efficient and Perfectly Unlinkable Sanitizable Signatures without Group Signatures. In *EuroPKI*, volume 8341 of *LNCS*, pages 12–30. Springer, 2013.
17. A. Buldas, P. Laud, and H. Lipmaa. Accountable certificate management using undeniable attestations. In *ACM Conference on Computer and Communications Security*, pages 9–17, 2000.
18. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, pages 61–76, 2002.
19. E.-C. Chang, C. L. Lim, and J. Xu. Short redactable signatures using random trees. In *CT-RSA*, pages 133–147, 2009.
20. H. de Meer, M. Liedel, H. C. Pöhls, J. Posegga, and K. Samelin. Indistinguishability of one-way accumulators. Technical Report MIP-1210, University of Passau, 2012.
21. H. de Meer, H. C. Pöhls, J. Posegga, and K. Samelin. On the relation between redactable and sanitizable signature schemes. In *ESSoS*, volume 8364 of *LNCS*, pages 113–130. Springer, 2014.
22. S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
23. S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM JoC*, 17:281–308, 1988.
24. J. Gong, H. Qian, and Y. Zhou. Fully-secure and practical sanitizable signatures. In *InsCrypt*, volume 6584 of *LNCS*, pages 300–317. Springer, 2011.

25. G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *Symposium on Principles of Database Systems*, PODS, pages 179–190, New York, USA, 2003. ACM.
26. S. Haber, Y. Hatano, Y. Honda, W. G. Horne, K. Miyazaki, T. Sander, S. Tezoku, and D. Yao. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In *ASIACCS*, pages 353–362, 2008.
27. C. Hanser and D. Slamanig. Blank digital signatures. In *AsiaCCS*, pages 95 – 106. ACM, 2013.
28. S. Hirose and H. Kuwakado. Redactable signature scheme for tree-structured data based on merkle tree. In *SECRYPT*, pages 313–320, 2013.
29. T. Izu, N. Kanaya, M. Takenaka, and T. Yoshioka. Piats: A partially sanitizable signature scheme. In Sihan Qing, Wenbo Mao, Javier López, and Guilin Wang, editors, *Information and Communications Security*, volume 3783 of *Lecture Notes in Computer Science*, pages 72–83. Springer Berlin Heidelberg, 2005.
30. T. Izu, M. Takenaka, J. Yajima, and T. Yoshioka. Integrity assurance for real-time video recording. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 651–655. IEEE, 2012.
31. R. Johnson, D. Molnar, D. Song, and D.Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262. Springer, Feb. 2002.
32. A. Kundu and E. Bertino. Privacy-preserving authentication of trees and graphs. *Int. J. Inf. Sec.*, 12(6):467–494, 2013.
33. J. Lai, X. Ding, and Y. Wu. Accountable trapdoor sanitizable signatures. In *ISPEC*, pages 117–131, 2013.
34. H. Lipmaa. Secure accumulators from euclidean rings without trusted setup. In *ACNS*, pages 224–240, 2012.
35. K. Miyazaki et al. Digitally Signed Document Sanitizing Scheme with Disclosure Condition Control. *IEICE Transactions*, 88-A(1):239–246, 2005.
36. K. Miyazaki and G. Hanaoka. Invisibly sanitizable digital signature scheme. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 91(1):392–402, 2008.
37. K. Miyazaki, G. Hanaoka, and H. Imai. Digitally signed document sanitizing scheme based on bilinear maps. In *ASIACCS*, pages 343–354. ACM, 2006.
38. K. Nyberg. Fast accumulated hashing. In *FSE*, pages 83–87, 1996.
39. H. C. Pöhls, S. Peters, K. Samelin, J. Posegga, and H. de Meer. Malleable signatures for resource constrained platforms. In *WISTP*, pages 18–33, 2013.
40. H. C. Pöhls, K. Samelin, H. de Meer, and J. Posegga. Flexible redactable signature schemes for trees - extended security model and construction. In *SECRYPT*, pages 113–125, 2012.
41. H. C. Pöhls, K. Samelin, and J. Posegga. Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In *ACNS*, volume 6715 of *LNCS*, pages 166–182. Springer-Verlag, 2011.
42. S. Rass and D. Slamanig. *Cryptography for Security and Privacy in Cloud Computing*. Artech House, 2013.
43. K. Samelin et al. On Structural Signatures for Tree Data Structures. In *ACNS*, volume 7341 of *LNCS*, pages 171–187. Springer, 2012.
44. K. Samelin et al. Redactable signatures for independent removal of structure and content. In *ISPEC*, volume 7232 of *LNCS*, pages 17–33. Springer-Verlag, 2012.
45. T. Sander. Efficient accumulators without trapdoor extended abstracts. In *ICICS*, pages 252–262, 1999.

46. D. Slamanig and S. Rass. Generalizations and extensions of redactable signatures with applications to electronic healthcare. In *Communications and Multimedia Security*, pages 201–213, 2010.
47. R. Steinfeld and L. Bull. Content extraction signatures. In *ICISC*. Springer Berlin / Heidelberg, 2001.
48. Z.-Y. Wu, C.-W. Hsueh, C.-Y. Tsai, F. Lai, H.-C. Lee, and Y. Chung. Redactable Signatures for Signed CDA Documents. *Journal of Medical Systems*, pages 1–14, December 2010.

## A Security Proofs of the Construction

We now show that our construction fulfills the given definitions. Namely, these are unforgeability, privacy, and transparency. We prove each property on its own.

**Our Scheme is Unforgeable.** If $\mathcal{AH}$ is strongly one-way, while the signature scheme $\Pi$ is unforgeable, our scheme is unforgeable.

**Proof.** Let $\mathcal{A}$ be an algorithm winning the unforgeability game. We can then use $\mathcal{A}$ in an algorithm $\mathcal{B}$ to either to forge the underlying signature scheme $\Pi$ or to break the strong one-wayness of $\mathcal{AH}$. Given the game in Fig. 5 we can derive that a forgery must fall in at least one of the two following cases, for at least one node $d$ in the tree:

- Type 1 Forgery: The value $d$ protected by $\sigma_s$ has never been signed by the signing oracle.
- Type 2 Forgery: The value $d$ protected by $\sigma_s$ has been signed, but $T^* \notin \text{span}_{\vdash}(T, \sigma, \texttt{ADM})$ for any tree $T$ signed by the signing oracle.

**Type 1 Forgery:** In the first case, we can use the forgery generated by $\mathcal{A}$ to create $\mathcal{B}$ which forges a signature. We construct $\mathcal{B}$ using $\mathcal{A}$ as follows:

1. $\mathcal{B}$ generates the key pair of $\mathcal{AH}$, i.e., $\texttt{pk} \leftarrow \mathsf{KeyGen}(1^\lambda)$. It passes $\texttt{pk}$ to $\mathcal{A}$. This is also true for $\texttt{pk}_S$ of the signature scheme to forge.
2. All queries to the signing oracle from $\mathcal{A}$ are genuinely answered with one exception: instead of signing digests itself, $\mathcal{B}$ asks it own signing oracle to generate the signature. Afterward, $\mathcal{B}$ returns the signature generated to $\mathcal{A}$.
3. Eventually, $\mathcal{A}$ outputs a pair $(T^*, \sigma^*)$. $\mathcal{B}$ looks for the message/signature pair $(m^*, \sigma_s^*)$ inside the transcript not queried to its own signing oracle, i.e., the accumulator value with the signature $\sigma_s^*$ of the root of $(T^*, \sigma^*)$. Hence, there exists a value not signed by $\mathcal{B}$'s signing oracle. This pair is then returned as $\mathcal{B}$'s own forgery attempt.

As every tree/signature pair was accepted as valid, but not signed by the signing oracle, $\mathcal{B}$ breaks the unforgeability of the signature algorithm. Here, we have a tight reduction for the first case.

**Type 2 Forgery:** In the case of a type 2 forgery, we can use $\mathcal{A}$ to construct $\mathcal{B}$, which breaks the strong one-wayness of the underlying accumulator. We construct $\mathcal{B}$ using $\mathcal{A}$ as follows:

1. $\mathcal{B}$ generates a key pair of a signature scheme $\Pi$.
2. It receives $\mathtt{pk}$ of $\mathcal{AH}$. Both public keys are forwarded to $\mathcal{A}$.
3. For every request to the signing oracle, $\mathcal{B}$ uses its hashing oracle to generate the witnesses and the accumulators. All other steps are genuinely performed. The signature is returned to $\mathcal{A}$.
4. Eventually, $\mathcal{A}$ outputs $(T^*, \sigma^*)$. Given the transcript of the simulation, $\mathcal{A}$ searches for a pair $(w^*, y^*)$ matching an accumulator $a$, while $y^*$ has not been queried to hashing oracle under $a$. Note, the root accumulator has been returned: otherwise, we have a type 1 forgery. $\mathcal{B}$ outputs $(a, w^*, y^*)$.

As every new element accepted as being part of the accumulator, while not been hashed by the hashing oracle, breaks the strong one-wayness of the accumulator, we have a tight reduction again.

**Our Scheme is Private.** If $\mathcal{AH}$ is indistinguishable our scheme is private. Note: the random numbers do not leak any information, as they are distributed uniformly and are not ordered. Hence, we do not need to take them into account.

**Proof.** Let $\mathcal{A}$ be an algorithm winning the privacy game. We can then use $\mathcal{A}$ in an algorithm $\mathcal{B}$ to break the indistinguishability of the accumulator $\mathcal{AH}$. We construct $\mathcal{B}$ using $\mathcal{A}$ as follows:

1. $\mathcal{B}$ generates a key pair of a signature scheme $\Pi$.
2. It receives $\mathtt{pk}$ of $\mathcal{AH}$. Both public keys are forwarded to $\mathcal{A}$.
3. For every request to the signing oracle, $\mathcal{B}$ produces the expanded trees given $\mathtt{ADM}$. Then, it uses its hashing-oracle to generate the accumulators, and then proceeds honestly as the original algorithm would do. Finally, it returns the generated signature $\sigma$ to $\mathcal{A}$.
4. For queries to the Left-or-Right oracle, $\mathcal{B}$ extracts the common elements to be accumulated for both trees — this set is denoted $\mathcal{S}$. Note, $\mathcal{S}$ may be empty. The additional elements for the first hash are denoted $\mathcal{R}_0$, and $\mathcal{R}_1$ for the second one. $\mathcal{B}$ now queries its own Left-or-Right oracle with $(\mathcal{S}, \mathcal{R}_0, \mathcal{R}_1)$ for each hash. The result is used as the accumulator and the witnesses required: $\mathcal{B}$ genuinely performs the rest of the signing algorithm and hands over the result to $\mathcal{A}$.
5. Eventually, $\mathcal{A}$ outputs its own guess $d$.
6. $\mathcal{B}$ outputs $d$ as its own guess.

As we only pass queries, $\mathcal{B}$ succeeds, whenever $\mathcal{A}$ succeeds.

**Our Construction is Transparent.** As our construction is private, it is only left to show that not doing any redaction — the additional case for transparency — does not affect the signature $\sigma$. This is obviously the case. Hence, the distributions are equal. Transparency follows.