FLEXIBLE REDACTABLE SIGNATURE SCHEMES FOR TREES
Extended Security Model and Construction

Henrich C. Pöhls[1,3*], Kai Samelin[2,3†], Hermann de Meer[2,3] and Joachim Posegga[1,3]
[1]*Chair of IT-Security, University of Passau, Germany*
[2]*Chair of Computer Networks and Computer Communication, University of Passau, Germany*
[3]*Institute of IT-Security and Security-Law (ISL), University of Passau, Germany*
{*hp,ks,jp*}*@sec.uni-passau.de, demeer@uni-passau.de*

Keywords: Redactable Signatures, Malleable Signatures, Trees

Abstract: At ISPEC '12, *Samelin* et al. show that the redactable signature scheme introduced at VLDB '08 by *Kundu* and *Bertino* does not always preserve the structural integrity of the tree signed. In particular, they show how redaction of non-leaves promotes descendants and allows a third party to add new edges to the signed tree. This alters the semantic meaning of the tree and is not acceptable in certain scenarios. We generalize the model, such that it offers the signer the flexibility to sign trees where every node is transparently redactable. This includes intermediates nodes, i.e, to allow redacting a hierarchy, but also the tree's root. We present a provably secure construction, where this possibility is given, while remaining under explicit control of the signer. Our security model is as strong as *Brzuska* et al.'s introduced at ACNS '10. We have implemented our secure construction and present a detailed performance analysis.

1 Introduction

Trees are commonly used to organize data. XML is just one of today's most prominent examples. To protect these documents against unauthorized modifications, digital signature algorithms like RSA (Rivest et al., 1983) can be used. Using these digital signatures schemes, two important properties of the data are protected: integrity of the data itself and verifiability of the signer and hence the data's origin. However, in certain scenarios, it is desirable to remove parts of a signed document without invalidating the protecting signature. Additionally, the remaining document shall still retain the integrity protection and origin verifiability offered by the signature.

This could be simply achieved by requesting a new signature from the signer with the parts removed before generating the new signature. While this roundtrip allows to satisfy the above requirements, the "digital document sanitization problem", as introduced in (Miyazaki et al., 2003), adds another requirement: the original signer shall not be involved again. This is useful in cases where the signer is not reachable or must not know that parts of a signed document are passed to third parties. Redactable signature schemes (RSS) have been designed to fulfill these needs, i.e., they allow that parts of a signed document can be removed without invalidating the signature. As standard signatures scheme like RSA, they allow to detect unauthorized changes of the signed data. In our case, the data is tree-structured, e.g., XML. Moreover, the tree's integrity needs to include the structure, i.e., the edges of the tree, as it carries information as well (Liu et al., 2009).

We found that existing RSSs for trees have two major shortcomings: (1) they differ in the integrity protection they offer for the tree's structure (structural integrity protection) (Kundu and Bertino, 2009). (2) they differ in the flexibility of redactions they allow (non-leaf redaction) (Brzuska et al., 2010a). We will explain the shortcomings of existing schemes, using the trees depicted in Fig. 1-4. Next, we will highlight the importance of each problems using illustrating examples:

Intermediate Node Redaction. Information stored in the tree's nodes might need to be redacted. Consider the tree depicted in Fig. 1, ignore the labels for now. To remove the leaf n_4, the node n_4 itself and the

*Is funded by BMBF (FKZ:13N10966) and ANR as part of the ReSCUeIT project (www.sichere-warenketten.de)

†Is supported by "Regionale Wettbewerbsfähigkeit und Beschäftigung", Bayern, 2007-2013 (EFRE) as part of the SECBIT project. (http://www.secbit.de)

edge $e_{3,4}$ is redacted from the tree $T = (V,E)$. Allowing leaf removal also allows to remove sub-trees, i.e., by consecutive removal of first n_4 and n_3 afterwards (Brzuska et al., 2010a). However, schemes only allowing for redaction of leaves fail to redact the data stored in n_3 only. In other words, we want that n_3 is redacted, while the nodes n_1,n_2 and n_4 are not subject to redaction. The resulting tree is depicted in Fig.2. To connect n_4 to the remaining tree, the third party requires to add a new edge, which was not present before, i.e., $e_{1,4}$. However, $e_{1,4}$ is in the *transitive closure* of T, as shown in Fig. 3. Hence, a RSS which is generally restricted to (consecutive) redaction of leaf nodes does not cater for all use cases. The scheme introduced by *Kundu* and *Bertino* does allow redaction of intermediate nodes, while also claiming that this flexibility is a useful property (Kundu and Bertino, 2009). However, this behavior may be problematic, as we will show next.

Structural Integrity Protection. Imagine the information stored in a tree T is a chart showing the employees' names as nodes and their position within the companies hierarchy (Liu et al., 2009). Hence, protecting structural integrity is equal to protecting the correctness of the employees' hierarchical positions in the given example. If one would only protect the ancestor relationship of nodes in T, one would only have a protection of all edges that are part of the transitive closure of T. This is depicted in Fig. 3. This allows a third party to add edges to the tree T' that were not present in T. *Samelin* et al. show that the scheme of *Kundu* and *Bertino* (KB-Scheme) is subject to this attack and name it "Level Promotion" (Samelin et al., 2012b). In our prior example, an employee can be "promoted". We will give a more detailed description of the attack on the KB-Scheme here and want to see it in the light of allowing flexibility. However, it motivates the need for structural integrity protection. A full description of their scheme is given in our extended notation in App. D. Their scheme builds upon the idea that a party who has all pre- and post-order traversal numbers (See Fig. 1) of all nodes contained in the tree T is always able to correctly reconstruct T. They argue that signing each node $n_i \in T$ along with both traversal numbers is enough to protect the tree's integrity.

Their verification checks all signatures on the nodes individually — with an additional step: A verifier has to check if all nodes are structured correctly using the traversal numbers (Kundu and Bertino, 2009). This leads to the problem that a verifier is not able to determine whether a given edge existed in the original tree T, just if it *could* have existed. Assume that a third party redacts n_3 from T, as depicted in Fig. 2. On verification the new edge $e_{1,4}$, which has not explicitly been present in the original tree T becomes valid. This implies, that the tree $T^{\mathcal{A}} = (\{n_1,n_2,n_4\},\{e_{1,2},e_{1,4}\})$ is valid.

To be more precise, let us compute the traversal numbers for the example tree in Fig. 1. The pre-order traversal of T will output $(1,2,3,4)$, while the post-order traversal will output $(4,3,2,1)$. To make their scheme not leaking occurred redaction, these traversal numbers are randomized in an order-preserving manner, which does not have an impact on the reconstruction algorithm. The randomization step may transform them into $(0.3,0.6,0.8,0.9)$ and $(0.7,0.4,0.2,0.1)$ resp. Hence, the node n_1 has a structural position of $\rho_1 = (0.3;0.7)$. For n_2, n_3 and n_4 this is done accordingly. We redact n_3, an intermediate node of n_1, and n_4. For the redacted tree in Fig. 2, the traversal-numbers are still in the correct order. Hence, the created signature verifies.

The impact of attacks on the structural integrity are significant, as the structure of a tree carries part of the document's information (Liu et al., 2009). One might argue that nesting of elements must adhere to a specific codified structure, i.e., XML-Schemata. Henceforth, attacks like level-promotions will be detected by any XML-Schema validator, if redactions are not valid. However, whenever an element can contain itself, like hierarchically structured lists of employees or hospital treatments composed of treatments, grandchildren can be promoted to direct children, without their traversal-number based validation-algorithm detecting it (Kundu and Bertino, 2009) and without breaking an XML-Schemata.

This destroys the structure of the tree and directly impacts on the semantics, maybe resulting in a different way on how a patient is treated in a hospital or impacting on the rights that one might believe an employee can have (Liu et al., 2009). Obviously, this is not acceptable in the generic case and could lead to several other attack vectors, similar to the ones XPath has introduced (Gottlob et al., 2003).

We conclude that the signer must *explicitly* sign the edges which can be used by the sanitizer rather then *implicitly* "signing" all of them. Currently, a signer is not able to control this behavior, neither in the KB-Scheme nor in other existing schemes (Samelin et al., 2012a).

State of the Art. Next, we provide an overview of the state of the art. The general concept of RSS were introduced at the same time by *Johnson* et al. in (Johnson et al., 2002) and *Steinfeld* and *Bull* in (Steinfeld and Bull, 2002). The latter approach

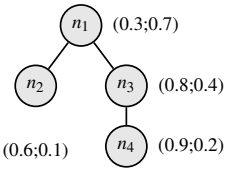

Figure 1: Original Tree T with Randomized Traversal Numbers

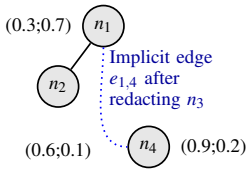

Figure 2: Intermediate Node Redaction T' following Kundu

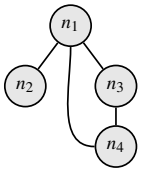

Figure 3: Transitive Closure of T implicitly adds $e_{1,4}$

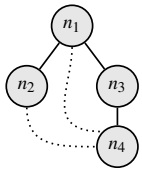

Figure 4: Allow Relocation ($e_{3,4}$) and Level-Promotion of n_4 ($e_{1,4}$). A dotted line denotes an explicit edge.

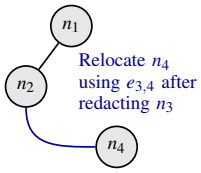

Figure 5: Redaction of n_3 and Re-location of n_4 using $e_{2,4}$

was named "content extraction signatures". A different but related concept are sanitizable signature schemes (Ateniese et al., 2005; Brzuska et al., 2009). Instead of only redaction of parts they allow to arbitrarily modify parts of the signed document. In this work, we concentrate on redactable signatures for trees. We take into account solutions that allow flexible redaction or offer full protection of structural- and content-integrity.

A recent work is *Ahn* et al. who cater for a special sub-case of redaction called "quoting" (Ahn et al., 2011). Quoting is not as flexible as redacting as it resembles "content extraction" by allowing quoting consecutive parts of a list. Hence, *Ahn* et al.'s approach is different and does not target trees.

Also recently, *Camacho* and *Hevia* have showed how to build more efficient transitive signatures for directed trees (Camacho and Hevia, 2012). This idea from *Rivest* and *Micali* (Micali and Rivest, 2002), however, focuses on how to authenticate single edges within a signed tree T.

Kundu and *Bertino* were the first to develop a RSS which addresses the specific needs of tree-structured documents (Kundu and Bertino, 2008; Kundu and Bertino, 2009). *Brzuska* et al. already broke the KB-Scheme and showed that it does not hide redactions, based on a probabilistic attack (Brzuska et al., 2010a). One may argue that *Kundu* and *Bertino* do not see their scheme in the context of RSS, since they just want to prohibit "leakage" due to structural information (Kundu and Bertino, 2009).

The solution by *Brzuska* et al. only allows redactions of leaves (Brzuska et al., 2010a). Another scheme for trees, by *Wu* et al. is also not flexible (Wu et al., 2010). Moreover, it relies on the *Merkle*-Hash-Tree-Technique with standard cryptographic hashes like SHA-512. Hence, their scheme does neither hide redactions nor preserve privacy (Kundu and Bertino, 2008; Kundu and Bertino, 2009).

Other schemes do not explicitly consider tree-structured data, even though they mention XML as an application: In (Pöhls et al., 2011) existing sanitizable and redactable signature schemes are integrated into XML Signatures, extending the work done in (Tan and Deng, 2009). Neither the approach of *Tan* et al., nor the work of *Pöhls* et al. do secure trees, since neither of them caters explicitly for the structure.

	Redaction of Non-leaves	Structural Integrity	Invisibility of Redaction
Kundu and *Bertino*	Yes, allowed	No, implicit level-promotion	No.
Brzuska et al.	No.	Yes.	Yes.
Wu et al.	No.	Yes.	No.
Comacho and *Hevia*	Yes, allowed	allows all transitive edges	different goal
Ahn et al.	No, only quoting	Yes.	Yes, stronger context-hiding [a]

[a] We discuss this in detail in Sect.2

Table 1: Existing RSS Schemes' Capabilities

Our Contribution and Outline. Our contribution is twofold and motivated by a lack of flexibility or integrity protection of many existing RSS for trees: Either no redaction of intermediate nodes is allowed or, if allowed, the structural integrity protection is relaxed to the transitive closure of the trees. As shown, protecting transitive closures, i.e., ancestor-descendant relations, is a weaker structural integrity protection which leads to semantic attacks which go unnoticed by the integrity protection.

Our first contribution is the rigid security model for a flexible RSS for trees offering full structural integrity protection. We follow *Brzuska* et al.'s security requirements for RSS for trees with leaf-only redaction (Brzuska et al., 2010a). In Sect. 2, we give precise, game-based definitions of the security properties: unforgeability, privacy, and transparency (Brzuska et al., 2010a). Our flexibility is allowing the signer to partially weaken the structural integrity protection at his leisure. Hence, the security requirements need to additionally capture the signer's flexibility to allow redaction of any node. This allows level promotions due to *re-locations* of specified sub-trees, which resembles the *implicit* possibility of previous schemes. In particular, our signing algorithm adds an additional edge to the tree to al-

low re-locations. A verifying party needs to decide which edge to use. This allows the sanitizer to maintain transparency after occurred modifications. On the other hand, the signer remains in charge when describing how an occurred redaction is hidden by re-locating the sub-tree. This leads to signer *explicitly* prohibiting the redaction of nodes *individually*, as the signer must *explicitly* sign an edge for re-location. Additionally, in our construction the signer controls the protection of the order of siblings. Hence, our scheme is capable of signing unordered trees. Our flexibility to redact an node of the tree does include in its generality the tree's root. Without loss of security, the signer can add an annotation to the root to prohibit redacting the roots by adapting the signing and verification algorithms. In particular, they must check, if the annotated root still exists.

Our second contribution is our secure construction presented in Sect. 3, including a performance measurement. Our construction is based upon a collision-resistant one-way accumulator (Benaloh and Mare, 1993; Barić and Pfitzmann, 1997) in combination with the *Merkle*-Hash-Tree-Technique (Merkle, 1989). Employing the *Merkle*-Hash-Tree-Technique enforces the protection of structural integrity. Hence, our scheme fulfills the defined security requirements using only standard cryptographic primitives.

We conclude our work in Sect. 4. The appendix contains the security proofs, the existing notations, the security model from (Brzuska et al., 2010a) and the KB-Scheme (Kundu and Bertino, 2009).

2 Extended Security Model for RSS for Trees

We will use the following notation throughout this paper: The root, denoted n_1, is the only node without a parent. Nodes are addressed by n_i. With c_i, we refer to all the content of node n_i, which is an additional information that might be associated with a node, i.e., data, element name and so forth. The first rigid model for secure RSS for trees was given in (Brzuska et al., 2010a). *Brzuska* et al. formalized the requirements for redactable signatures for tree-structured documents. Their model only allows removing leaves; sequentially running their leaf-cutting algorithm only removes complete sub-trees. However, the model is restrictive with the respect, that it only allows redacting leaves of the tree. (Brzuska et al., 2010a) We restate their model and review it in App. C. We will now rework the *Brzuska* et al.'s model to securely allow the possibility to redact any node. After we have stated our new security model, we will shortly compare the new model to *Brzuska* et al.'s and *Ahn* et al.'s (Ahn et al., 2012; Brzuska et al., 2010a). Keep in mind, that our flexibility allows to work on ordered and un-ordered trees, and generally we also allow redacting a tree's root.

Flexible Redactable Signature Scheme for Trees. A RSS $\mathcal{RSS}_T$ for trees consists of the four efficient (PPT)[3] algorithms: $\mathcal{RSS}_T := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Modify})$. Note, all algorithms may output $\perp$ in case of an error.

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

Sign. On input of the secret key sk, the tree T and ADM the algorithm Sign outputs the signature σ_T. ADM controls what changes by Modify are admissible. In detail, ADM is the set containing all signed edges, including the ones where a sub-tree can be re-located to. In particular, $(n_i, n_j) \in \text{ADM}$, iff the edge (n_i, n_j) is to be signed. These edges cannot be derived from T alone. For simplicity, we assume that ADM is always correctly derivable from (T, σ_T). The output: $(T, \sigma_T, \text{ADM}) \leftarrow \mathsf{Sign}(sk, T, \text{ADM})$.

Verify. On input of the public key pk, the tree T and the signature σ_T the algorithm Verify outputs a bit $d \in \{0,1\}$ indicating the correctness of the signature σ_T, w.r.t. pk, protecting the tree T. The output: $d \leftarrow \mathsf{Verify}(pk, T, \sigma_T)$.

Modify. The algorithm Modify takes as input the signer's public key pk, the tree T, the signature σ_T and ADM of T, and an instruction MOD. MOD contains the actual change to be made: redact a leaf n_i, relocate a sub-tree T_ψ, distribute a sub-tree T_υ without the original root, or prohibit relocating a sub-tree T_ϖ. A modification of T w.r.t. MOD is denoted as $T \otimes \text{MOD}$.

Apart from potentially changing T, the old ADM must be adjusted: In particular, if a node n_i is redacted, the edge to its father needs to be removed as well. Moreover, if there exists a sub-tree which could be relocated under the redacted node, the corresponding edges need to be removed from ADM as well. A modification of ADM w.r.t. MOD will be denoted as $\text{ADM} \otimes \text{MOD}$. The alteration of ADM is crucial to maintain privacy and transparency. Note, running Modify multiple times is the same as a MOD containing more than one change to be made. The output: $(T', \sigma'_T, \text{ADM}') \leftarrow \mathsf{Modify}(pk, T, \sigma_T, \text{ADM}, \text{MOD})$.

[3] probabilistic polynomial-time (PPT)

Correctness. Genuinely signed or signed and modified trees are considered valid by the verification algorithm. We use $span_{\vdash}(T, \text{ADM})$ to denote all valid trees that can be generated from a signed tree T by running Modify never, once or more times, with a MOD which respects ADM. Hence, we require all elements of $span_{\vdash}(T, \text{ADM})$ have valid signatures, iff T has a valid signature. Note: Prohibiting additional *relocations* is allowed in our scheme. We will always explicitly denote ADM in our algorithms for clarity.

The Extended Security Model. As discussed before, *Ahn* et al.'s framework has a stronger notation for privacy than *Brzuska* et al. Our scheme is as secure as *Brzuska* et al.'s. We extend this model to cater for the flexibility of intermediate node redaction and re-locations, so the security properties must hold also when the signer is given the new freedom to allow any node being removed.

1. *Unforgeability:* No one should be able to compute a valid signature on a tree T^* for pk outside $span_{\vdash}(T, \text{ADM})$ without access to the corresponding secret key sk. This is analogous to the standard unforgeability requirement for signature schemes, as already noted in (Brzuska et al., 2010a). A scheme *RSS* is unforgeable, iff for any efficient (PPT) adversary $\mathcal{A}$, the probability that the game depicted in Fig. 6 returns 1, is negligible (as a function of λ). In the game the attacker is given access to a signature generating oracle $\mathsf{Sign}(sk, \cdot, \cdot)$ and the public key pk, but not the secret key. The attacker wins if he has a valid signature for T^*, which is a tree for which he has never queried the oracle directly, nor can he generate T^* by modifying a previously queried signed tree.
2. *Privacy:* No one should be able to gain any knowledge about the unmodified tree without having access to it. This is similar to the standard indistinguishability notation for encryption schemes (Brzuska et al., 2010a). A scheme *RSS* is private, iff for any efficient (PPT) adversary $\mathcal{A}$, the probability that the game shown in Fig. 7 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ). In the game the attacker is given a signature generating oracle and the public key. He controls two inputs to the LoRModify oracle (Fig. 8) and also how they need to be modified to result in the same tree. The oracle modifies both of them into the same tree and outputs one of them to the attacker. The attacker needs to identify the used input to win. So the attacker controls all the inputs $T_{j,0}, \text{ADM}_{j,0}, \text{MOD}_{j,0}, T_{j,1}, \text{ADM}_{j,1}, \text{MOD}_{j,1}$, but need to guess which of the (now modified) trees is returned. Note, that the two inputs need to be modified such that they are equal w.r.t. T, σ and also ADM.
3. *Transparency:* A party who receives a signed tree T cannot tell whether he received a freshly signed tree or a tree which has been created via Modify (Brzuska et al., 2010a). We say that a scheme *RSS* is transparent, if for any efficient (PPT) adversary $\mathcal{A}$, the probability that the game shown in Fig. 9 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ). In the game for transparency, the attacker has access to the public key and a signature generation oracle. He controls the input to the ModifyOrSign oracle (Fig. 10). Hence, he can choose the original tree T with admissible modifications ADM and by MOD also the modified tree. Note, MOD can contain several modification instructions. To win, the attacker has to guess if the signed outputted T' was created through the modification algorithm from a signed T ($b = 0$) or through modifying T and ADM before signing them ($b = 1$).

Experiment $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathsf{RSS}_T}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(T^*, \sigma_T^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot,\cdot)}(pk)$
let $i = 1, 2, \ldots, q$ index the queries
return 1 iff
$\mathsf{Verify}(pk, T^*, \sigma_T^*) = 1$ and
for all $1 \le i \le q$, $T^* \notin span_{\vdash}(T_i, \text{ADM}_i)$

Figure 6: Game for Unforgeability

Experiment $\mathsf{Privacy}_{\mathcal{A}}^{\mathsf{RSS}_T}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot,\cdot),\mathsf{LoRModify}(\cdot,\cdot,\cdot,\cdot,\cdot,sk,b)}(pk)$
return 1 iff $b = d$

Figure 7: Game for Privacy

Oracle $\mathsf{LoRModify}(T_{j,0}, \text{ADM}_{j,0}, \text{MOD}_{j,0}, T_{j,1}, \text{ADM}_{j,1}, \text{MOD}_{j,1}, sk, b)$
if $\text{MOD}_{j,0}(T_{j,0}) \neq \text{MOD}_{j,1}(T_{j,1})$ return $\perp$
$(T_{j,0}, \sigma_{T,0}, \text{ADM}_{j,0}) \leftarrow \mathsf{Sign}(sk, T_{j,0}, \text{ADM}_{j,0})$
$(T_{j,1}, \sigma_{T,1}, \text{ADM}_{j,1}) \leftarrow \mathsf{Sign}(sk, T_{j,1}, \text{ADM}_{j,1})$
$(T'_{j,0}, \sigma'_{T,0}, \text{ADM}'_{j,0}) \leftarrow \mathsf{Modify}(pk, T_{j,0}, \sigma_{T,0}, \text{ADM}_{j,0}, \text{MOD}_{j,0})$
$(T'_{j,1}, \sigma'_{T,1}, \text{ADM}'_{j,1}) \leftarrow \mathsf{Modify}(pk, T_{j,1}, \sigma_{T,1}, \text{ADM}_{j,1}, \text{MOD}_{j,1})$
if $\text{ADM}'_{j,0} \neq \text{ADM}'_{j,1}$ abort returning $\perp$
return $(T'_{j,b}, \sigma'_{T,b}, \text{ADM}'_{j,b})$

Figure 8: LoRModify Oracle (from Privacy)

Experiment $\mathsf{Transparency}^{\mathsf{RSS}_T}_{\mathcal{A}}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot,\cdot),\mathsf{ModifyOrSign}(\cdot,\cdot,\cdot,sk,b)}(pk)$
return 1 iff $b = d$

Figure 9: Game for Transparency

Oracle $\mathsf{ModifyOrSign}(T, \text{ADM}, \text{MOD}, sk, b)$
if $\text{MOD} \notin \text{ADM}$, return $\perp$
if $b = 0$: $(T, \sigma_T, \text{ADM}) \leftarrow \mathsf{Sign}(sk, T, \text{ADM})$,
$(T', \sigma'_T, \text{ADM}') \leftarrow \mathsf{Modify}(pk, T, \sigma_T, \text{ADM}, \text{MOD})$
if $b = 1$: $T' \leftarrow T \otimes \text{MOD}$, $\text{ADM}' \leftarrow \text{ADM} \otimes \text{MOD}$
$(T', \sigma'_T, \text{ADM}') \leftarrow \mathsf{Sign}(sk, T', \text{ADM}')$
return $(T', \sigma'_T, \text{ADM}')$.

Figure 10: ModifyOrSign Oracle (from Transparency)

Separation of Security Properties. The implications and separations of *Brzuska* et al. do not change: Unforgeability is independent; Transparency $\Rightarrow$ Privacy; Privacy $\not\Rightarrow$ Transparency. We omit the proofs in this paper; they are essentially the same as given in (Brzuska et al., 2010a).

Security of our Model. Our security model offers the same security as *Brzuska* et al.'s: unforgeability, privacy and transparency notations (Brzuska et al., 2010a).

The security model of *Ahn* et al.'s important work (Ahn et al., 2012) introduces "strong context-hiding" as a very strong privacy notation. Compared to *Brzuska* et al.'s privacy notation, context-hiding will even hide the fact "whether it $[(T', \sigma', \text{ADM}')]$ was derived from a given signed message" when the attacker has access to a real original message and signature, i.e., (T, σ, ADM) and the secret key sk. This is considered a very strong privacy property by (Boneh and Freeman, 2011) and we do not achieve this, as we do not achieve unlinkability (Brzuska et al., 2010b). However, in most use cases this strong privacy notation is not needed, since the existing side-channel information already links two signatures, i.e., non-personal governmental datasets being released to foster Open Government initiatives.

Redacting any Node. To show the full flexibility of allowing any node to be redactable we do treat the tree's root as a redactable node. Note, redacting the root, as shown in Fig. 11 (3c), is possible. In the example (3c) n_2 transparently becomes the new root. However, in general redacting, the root might leave one with a forest of trees, i.e., two or more unconnected sub-trees. Each sub-tree is than a valid signed sub-tree, however the connection between the sub-trees is lost. Iff the signer provided re-location edges that allow to connect the sub-trees into one tree then this is a possible option after having redacted the root.

Adding signer control, to prohibit this is straight forward: During signing we annotate n_1 as root and additionally indicate that redacting the root is prohibited. Complementary, the verify algorithm will check for that indicator and, iff present, it will verify, if one node with the annotation is present in the root node received.

3 Our New Secure Flexible Scheme

Merkle-Hash-Tree ($\mathcal{MH}$). Our construction follows the ideas of the *Merkle*-Hash-Tree (Merkle, 1989): We use the following notation for the recursively constructed extended version, which works on arbitrary trees with content. We denote a concatenation of two strings x, y with $x||y$. The extended *Merkle*-Hash $\mathcal{MH}$ 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 cryptographic hash function like SHA-512, c_x the content of the node x, x_i a child of x, and n the number of children of x. $\mathcal{MH}(n_1)$'s output, called root hash, depends on all nodes and on the *right order* of the siblings. Hence, signing the root hash 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 will most likely cause a different digest value. One may argue that annotating an unordered sub-tree is sufficient. However, this does not allow to rearrange items and hence enforces a given order, which may not be wanted in certain use-cases, e.g., if invoices are signed. Hence, an unsorted list or tree should be signed and verified as such. A more detailed analysis of the *Merkle*-Hash-Tree is given in (Kundu and Bertino, 2009), which also gives an introduction on the possible inference attacks on non-private schemes.

Accumulating Hash-Functions ($\mathcal{AH}$). One-way accumulators have been introduced by in (Benaloh and Mare, 1993). The basic idea is to construct a strongly one-way family of functions $\mathcal{AH}_K$, where $\forall \mathcal{AH}_k \in \mathcal{AH}_K : \mathcal{X}_k \times \mathcal{Y}_k \rightarrow \mathcal{X}_k$. We just need the basic operations of an accumulator, e.g., no dynamic updates or revokation techniques are required. A basic accumulator consists of three efficient algorithms, i.e., $\mathcal{AH} := \{\mathsf{KeyGen}, \mathsf{Hash}, \mathsf{Check}\}$:

KeyGen. Outputs the system parameters prm on input of a security parameter λ, i.e., $prm \leftarrow$

$\mathsf{KeyGen}(1^\lambda)$

Hash. Outputs the accumulated digest d along with the set of witnesses $\mathcal{W} = \{w_1, \ldots, w_n\}$, i.e., $(d, \mathcal{W}) \leftarrow \mathsf{Hash}(prm, I)$, on input of a set of to-be-digested items $I = \{v_1, \ldots, v_n\}$ and the parameters prm. The accumulatation of $I = \{v_1; \ldots; v_n\}$ is denoted as $\mathcal{AH}_k(prm, \{v_1; \ldots; v_n\})$, where $k \in K$. Each witnesses w_ℓ in $\mathcal{W}$ allows to prove the membership of the corresponding value v_ℓ.

Check. Outputs a bit $d \in \{0,1\}$ indicating if a given value v_i was accumulated into the digest d with respect to prm and a witness w_i. In particular, $b \leftarrow \mathsf{Check}(prm, v_i, d, w_i)$

In some papers, the witness generation is done by an algorithm Proof. We will use this algorithm in the appendix to describe our requirements formally. However, the above notation allows to shorten the algorithmic description. We require the usual soundness requirements (Barić and Pfitzmann, 1997) to hold, while the concrete instantiation of an accumulator must be strongly one-way (Barić and Pfitzmann, 1997). In particular, an outsider should not be able to guess members or to generate membership proofs. To achieve transparency, we also require that the accumulator does not leak how many additional members a digest has. The formal descriptions of our needs are relegated to App. B. Additional information about accumulators can be found in (Barić and Pfitzmann, 1997; Benaloh and Mare, 1993; Camenisch and Lysyanskaya, 2002). To maintain transparency, we require that any output of $\mathcal{AH}$ is distributed uniformly over the co-domain of the hash-function. An accumulator not fulfilling these requirements has been proposed by *Nyberg* in (Nyberg, 1996); the underlying *Bloom*-Filter can be attacked by probabilistic methods and therefore leaks the amount of members. This is not acceptable. To prohibit recalculations of a digest, we require a nonce as the seed. This has already been proposed in (Nyberg, 1996) and (Barić and Pfitzmann, 1997). The idea to use accumulating hashes has already been proposed by *Kundu* and *Bertino* in (Kundu and Bertino, 2008). However, they state that accumulators are not able to achieve the desired functionality. We will show that they are sufficient by giving a concrete construction.

The Construction. We want to allow explicit relocation of sub-trees. If a non-leaf is subject to redaction, all sub-trees of the node need to be relocated. If this is possible and what their new ancestor will be must be under the sole control of the signer. We will first sketch our solution, and give a concrete instantiation and the algorithms afterwards. Our re-location definition does not require to delete the intermediate node. This behaviour will be discussed, after the construction has been introduced.

Sketch. Our solution requires that the signer replicates all re-locatable nodes and the underlying sub-trees to all the locations where a sanitizer is allowed to relocate the sub-tree to. The replicas of the nodes are implicitly used just to produce the re-locatable edges as our algorithms all work on nodes. Each additional edge is also noted in ADM. In Fig. 11(1+2), the dotted area corresponds to the sub-tree n_3 and n_4 under the re-locatable n_3. The sub-tree n_3 and n_4 must be re-located as a whole. The dashed curved edge $e_{1,3}$ corresponds to such an additional edge contained in ADM to indicate the allowed re-location of n_3 as direct child of n_1. All algorithms work on nodes not on edges, hence always imagine one builds all allowed re-locations by replicating nodes before one runs an algorithm like MODIFY, as depicted in Fig. 11(1).

We allow to re-locate a re-locatable sub-tree without redaction of nodes. We prohibit simple copy attacks, i.e., leaving a relocated sub-tree T_ω in two locations, because each node n_i gets an associated unique nonce r_i. The whole tree gets signed as in the standard *Merkle*-Hash-Tree, with one notable exception: Instead of using a standard hash like SHA-512, we use an accumulator which allows the sanitizer to remove values without changing the digest value. To remove parts the sanitizer is removing the redacted elements and no longer provides the corresponding witnesses of the redacted elements. The accumulators also allows us to sign unordered trees; 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 allow distribution of sub-trees without the root, we sign each node's *Merkle*-Hash individually using a standard signature scheme. As said, redaction is therefore a simple removal of the nodes and the corresponding witnesses. Re-location is similar: Apply the necessary changes to T. Additionally, a sanitizer can prohibit consecutive re-locations, this could be seen equal to consecutive sanitization control (Miyazaki et al., 2005). To prohibit further re-location one removes the corresponding witnesses. This implies that ADM is adjusted accordingly.

Verification: For a node x 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 for each node. A seed is used to prohibit simple recalculation attacks (Barić and Pfitzmann, 1997). To sign the ordering between siblings, we sign the "left-of" relation, as already used and proposed in (Brzuska et al., 2010a), (Chang et al.,

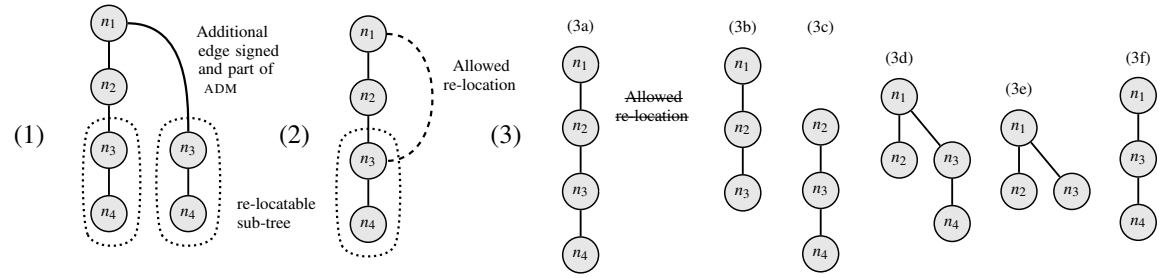

Figure 11: (1) Expanded tree with duplicates for all possible re-locations of sub-trees, (2) Tree with allowed Level-Promotion of n_3, (3) Examples of valid trees with 3 or 4 nodes after: (3a) redact re-locatability, (3b) redact n_4, (3c) redact root, (3d) relocate-only n_3, (3e) re-locate n_3 and redact n_4, (3f) redact n_2 and level-promote n_3

2009) and (Samelin et al., 2012b).

The Algorithmic Description. We assume that witnesses are generated, distributed and used to verify memberships, and, of course, removed from distribution, if the corresponding values are removed. This does not introduce any security problems, since the sanitizer could give away the original tree anyway. How the witnesses are generated depends on the actual accumulator used and is therefore omitted.

KeyGen(λ):
//Choose a suitable accumulator
Choose $\mathcal{AH}_k \in \mathcal{AH}_K$
Generate $prm \leftarrow \mathsf{AHKeyGen}_k(\lambda)$
//Choose an unforgeable signature scheme Π
Choose Π and set $(pk_S, sk_S) \leftarrow \Pi.\mathsf{AKeyGen}(\lambda)$
//Return all generated material
return $(pk = (pk_S, prm, \mathcal{AH}_k, \Pi), sk = (sk_S))$

Expand(T, ADM):
For all edges $e_i \in \text{ADM}$
Replicate the sub-tree underneath the
node addressed by e_i
to the designated position
this must be done bottom-up
Note: This implies that r_i are copied as well.
Return this expanded tree, denoted as Ω

Sign(sk, T, ADM):
For each node $n_i \in T$:
$r_i \xleftarrow{\$} \{0,1\}^\lambda$
If r_i has already been drawn, draw again
Expanded tree $\Omega \leftarrow \mathsf{Expand}(T, \text{ADM})$
For each node $x \in \Omega$:
Draw a random seed $s_x \xleftarrow{\$} \{0,1\}^\lambda$
Let x_i denote a children of x
If the tree is un-ordered:
$d_x = \mathcal{MH}(x) \leftarrow \mathcal{AH}_k(prm, \{s_x; c_x || r_x; \mathcal{MH}(x_1); \ldots; \mathcal{MH}(x_n)\})$
$\sigma_x \leftarrow \Pi.\mathsf{ASign}(sk_S, d_x || \text{unordered})$
Else: //ordered tree
$d_x = \mathcal{MH}(x) \leftarrow \mathcal{AH}_k(prm, \{s_x; c_x || r_x; \mathcal{MH}(x_1); \ldots; \mathcal{MH}(x_n); \Xi_x\})$
// Build Ξ_x, the set of
// all "left-of" relations of x_i:
$\Xi_x = \{r_i || r_j \mid 0 < i < j \leq n\}$
$\sigma_x \leftarrow \Pi.\mathsf{ASign}(sk_S, d_x || \text{ordered})$
// Build list of all witnesses:
Let $\mathcal{W} = \{w_i | n_i \in \Omega\}$.
return $\sigma_T = (\{\sigma_i\}_{n_i \in \Omega}, \mathcal{W}, \text{ADM})$

Modify(pk, T, σ_T, ADM, MOD):
use Verify to verify the tree T
Expanded tree $\Omega \leftarrow \mathsf{Expand}(\mathsf{T}, \text{ADM})$
Case 1: Redact node n_l:
//1. remove **all** n_l (incl. replicas) from Ω:
Set $\Omega' \leftarrow \Omega \setminus n_l$
//2. remove **the** node n_l from T:
Set $T' \leftarrow T \setminus n_l$
return $\sigma'_T = (T', \{\sigma_i\}_{n_i \in \Omega'}, \{w_i\}_{n_i \in \Omega'}, \text{ADM})$
Case 2: Share sub-tree $T_\mathfrak{v}$, where $n_1 \notin T_\mathfrak{v}$:
return $\sigma'_T = (T_\mathfrak{v}, \{\sigma_i\}_{n_i \in T_\mathfrak{v}}, \{w_i\}_{n_i \in T_\mathfrak{v}}, \text{ADM})$
Case 3: Re-locate T_ψ:
Set $T' \leftarrow \text{MOD}(T)$
return $\sigma'_T = (T', \{\sigma_i\}_{n_i \in \Omega}, \{w_i\}_{n_i \in \Omega}, \text{ADM})$
Case 4: Remove re-location edge e_l:
Set $\text{ADM}' \leftarrow \text{ADM} \setminus e_l$
Expanded tree $\Omega' \leftarrow \mathsf{Expand}(T, \text{ADM}')$
//Note: This expansion is done with the modified ADM$'$.
return $\sigma'_T = (T, \{\sigma_i\}_{n_i \in \Omega'}, \{w_i\}_{n_i \in \Omega'}, \text{ADM}')$

Verify(pk, T, σ_T):
Check if each $r_i \in T$ is unique.
For each node $x \in T$:
Let the value protected by σ_x be d_x
Let $d \leftarrow \mathsf{Check}(prm, c_x || r_x, d_x, w_x)$
If $d = 0$, return 0
For all children c_i of c do:
let the value protected by σ_{c_i} be d_c
//Note: checks if children are signed
Let $d \leftarrow \mathsf{Check}(prm, d_c, d_x)$
If $d = 0$, return 0
If $\sigma_x = d_x || \text{ordered}$:

	Generation of σ			Verification of σ		
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 2: Median Runtime; in ms

//Is every "left-of"-relation signed?
//Note: just linearly many checks
For all $0 < i < n$:
$d \leftarrow \mathsf{Check}(prm, r_i || r_{i+1}, d_x, w_{x,x+1})$
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 patients record. A third party can prohibit consecutive re-locations by removing the associated witnesses from distribution. An example of an resulting tree is depicted in Fig.11(3a). Note, the algorithm Modify actually works by removing the duplicated nodes, and hence removes the allowed edge $e_{1,3}$.

Runtime Complexity. For generation of σ_T, the sibling's order requires $\frac{n(n-1)}{2}$ hashing steps. All other steps require to touch the nodes only once. Consequently, the runtime approximation of our signing algorithm is linear in the number of nodes, while being quadratic in the number of siblings. Redacting is just removing values. Verification is $O(|V|)$, since a verifier has to check, if each digest is contained in its parent's digest and if the content has been digested. Checking the order of siblings can be done in linear time due to the transitive behaviour. We have implemented our scheme to demonstrate the practical usability. As the accumulator, we have choosen the original construction (Benaloh and Mare, 1993). 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). Source code will be made available upon request. We took the median of 10 runs. We measured trees with unordered siblings and one with ordered siblings. Time for generation of keys for the hash is included. We excluded the time for creating the required key pairs; it becomes negligible in terms of the performance for large n. On digest calculation we store all intermediate results in RAM to avoid any disk access impact.

As shown, our construction is useable, but becomes slow for large branching factors. 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., (Brzuska et al., 2010a) and (Chang et al., 2009), have the same complexity and therefore just differ by a constant factor, but do not provide a performance analysis on real data.

Security of the Construction. Our scheme is unforgeable, private and transparent. Assuming $\mathcal{AH}$ is strongly one-way, and the signature scheme Π is UNF-CMA, our scheme is unforgeable, while $\mathcal{AH}$ always outputs uniformly distributed digests and witnesses, our scheme is transparent and also private. Note, it is enough to show that Transparency and Unforgeability hold because Transparency $\Longrightarrow$ Privacy (Brzuska et al., 2010a). The formal proofs are relegated to App. A for readability.

4 Conclusion

The lack of flexibility for redacting any node of a tree and the lack of signer control motivated the construction of a new RSS. The scheme can sign ordered and unordered trees. It allows the sanitizer to redact non-leaf nodes and keeps the redaction transparent by allowing level-promotions, but level promotions are under the signer's sole control. Keeping the signer in control gives him the decision for which intermediate nodes he wants to weaken the structural integrity protection and allow a re-location, but allows transparency.

Furthermore, our construction is the first which allows a signer to explicitly sign ordered and unordered trees. How to sign trees where both, ordered and unordered siblings, are present is still an open problem. We allow re-locations without redaction so that a signer can allow a sanitizer to redact the hierarchy from a sub-tree which contains hierarchically structured, but otherwise equal, nodes. In these cases, our scheme allows redacting just the structure. Such a violation of structural integrity requires explicit confirmation by the signer.

Allowing the flexibility required us to give an extended security model. Finally, we have proven our scheme using the extended security model. The performance analysis demonstrates that the scheme is considerably slower than SHA-512. It therefore remains an open problem how to construct transparent schemes with an overhead of $O(n)$ and how to mix ordered and unordered trees.

REFERENCES

Ahn, J. H., Boneh, D., Camenisch, J., Hohenberger, S., Shelat, A., and Waters, B. (2011). Computing on authenticated data. Cryptology ePrint Archive, Report 2011/096. http://eprint.iacr.org/.

Ahn, J. H., Boneh, D., Camenisch, J., Hohenberger, S., Shelat, A., and Waters, B. (2012). Computing on authenticated data. In Cramer, R., editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 1–20. Springer.

Ateniese, G., Chou, D. H., de Medeiros, B., and Tsudik, G. (2005). Sanitizable Signatures. In *ESORICS*, pages 159–177.

Barić, N. and Pfitzmann, B. (1997). Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, pages 480–494.

Benaloh, J. and Mare, M. D. (1993). One-way accumulators: A decentralized alternative to digital signatures. pages 274–285. Springer-Verlag.

Boneh, D. and Freeman, D. M. (2011). Homomorphic signatures for polynomial functions. In *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 149–168.

Boneh, D., Gentry, C., Lynn, B., and Shacham, H. (2003). Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *EUROCRYPT*, pages 416–432.

Brzuska, C., Busch, H., Dagdelen, O., Fischlin, M., Franz, M., Katzenbeisser, S., Manulis, M., Onete, C., Peter, A., Poettering, B., and Schröder, D. (2010a). Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security*, ACNS'10, pages 87–104. Springer.

Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., and Volk, F. (2009). Security of Sanitizable Signatures Revisited. In *Proc. of PKC 2009*, pages 317–336. Springer.

Brzuska, C., Fischlin, M., Lehmann, A., and Schröder, D. (2010b). Unlinkability of Sanitizable Signatures. In *Public Key Cryptography*, pages 444–461.

Camacho, P. and Hevia, A. (2012). Short transitive signatures for directed trees. In Dunkelman, O., editor, *Topics in Cryptology – CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*, pages 35–50. Springer Berlin / Heidelberg.

Camenisch, J. and Lysyanskaya, A. (2002). Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, pages 61–76.

Chang, E.-C., Lim, C. L., and Xu, J. (2009). Short Redactable Signatures Using Random Trees. In *Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, CT-RSA '09, pages 133–147, Berlin, Heidelberg. Springer-Verlag.

Gottlob, G., Koch, C., and Pichler, R. (2003). The complexity of XPath query evaluation. In *Proceedings of the 22nd Symposium on Principles of Database Systems*, PODS, pages 179–190, New York, USA. ACM.

Johnson, R., Molnar, D., Song, D., and D.Wagner (2002). Homomorphic signature schemes. In *Proceedings of the RSA Security Conference - Cryptographers Track*, pages 244–262. Springer.

Kundu, A. and Bertino, E. (2008). Structural Signatures for Tree Data Structures. In *Proc. of PVLDB 2008*, New Zealand. ACM.

Kundu, A. and Bertino, E. (2009). CERIAS Tech Report 2009-1 Leakage-Free Integrity Assurance for Tree Data Structures.

Liu, B., Lu, J., and Yip, J. (2009). XML data integrity based on concatenated hash function. *International Journal of Computer Science and Information Security*, 1(1).

Merkle, R. C. (1989). A certified digital signature. In *CRYPTO*, pages 218–238.

Micali, S. and Rivest, R. L. (2002). Transitive signature schemes. In Preneel, B., editor, *CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 236–243. Springer.

Miyazaki, K., Iwamura, M., Matsumoto, T., Sasaki, R., Yoshiura, H., Tezuka, S., and Imai, H. (2005). Digitally Signed Document Sanitizing Scheme with Disclosure Condition Control. *IEICE Transactions*, 88-A(1):239–246.

Miyazaki, K., Susaki, S., Iwamura, M., Matsumoto, T., Sasaki, R., and Yoshiura, H. (2003). Digital documents sanitizing problem. Technical Report ISEC2003-20, IEICE.

Nyberg, K. (1996). Fast accumulated hashing. In *FSE*, pages 83–87.

Pöhls, H. C., Samelin, K., and Posegga, J. (2011). Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In *Applied Cryptography and Network Security, 9th International Conference*, volume 6715 of *LNCS*, pages 166–182. Springer-Verlag.

Rivest, R. L., Shamir, A., and Adleman, L. (1983). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99.

Samelin, K., Pöhls, H. C., Bilzhause, A., Posegga, J., and de Meer, H. (2012a). On Structural Signatures for Tree Data Structures. In *Applied Cryptography and Network Security, 10th International Conference*, volume 7341 of *LNCS*, pages 171–187. Springer-Verlag.

Samelin, K., Pöhls, H. C., Bilzhause, A., Posegga, J., and de Meer, H. (2012b). Redactable signatures for independent removal of structure and content. In *ISPEC*, volume 7232 of *LNCS*, pages 17–33. Springer-Verlag.

Steinfeld, R. and Bull, L. (2002). Content extraction signatures. In *Information Security and Cryptology - ICISC 2001: 4th International Conference*. Springer Berlin / Heidelberg.

Tan, K. W. and Deng, R. H. (2009). Applying Sanitizable Signature to Web-Service-Enabled Business Processes: Going Beyond Integrity Protection. In *ICWS*, pages 67–74.

Wu, Z.-Y., Hsueh, C.-W., Tsai, C.-Y., Lai, F., Lee, H.-C., and Chung, Y. (2010). Redactable Signatures for Signed CDA Documents. *Journal of Medical Systems*, pages 1–14.

A Security Proofs of the Construction

Our changes to the security model do not affect the implications and separations as presented in (Brzuska et al., 2010a). Hence, unforgeability is independent, while transparency $\Rightarrow$ privacy and privacy $\not\Rightarrow$ transparency (Brzuska et al., 2010a). The proofs are essentially the same as already given in (Brzuska et al., 2010a). Following from that it is sufficient to show that transparency and unforgeability hold to show that our scheme are secure. We will show this for each property on its own.

The Construction is unforgeable If $\mathcal{AH}$ is strongly one-way, while the signature scheme Π is unforgeable, our scheme is unforgeable.

Proof. Let $\mathcal{A}_{unf}$ be an algorithm winning our unforgeability game. We can then use $\mathcal{A}_{unf}$ to forge the underlying signature scheme or to calculate membership proofs. Hence, our scheme's security relies upon the security of the signature scheme and $\mathcal{AH}$. Given the game in Fig. 6 we can derive that a forgery must fall in at least one of the three cases, for at least one node in the tree:

Type 1 Forgery: The value d protected by σ_T has never been queried by $\mathcal{A}_{unf}$ to O^{Sign} Type 2 Forgery: The value d protected by σ_T has been queried by $\mathcal{A}_{unf}$ to O^{Sign}, but $T^* \notin \text{span}_{\vdash}(T, \text{ADM})$; so the tree T^* with valid signature σ_T is not in the transitive closure of T. This case has to be divided as well:

Type 2a Forgery: $T \notin \text{span}_{\vdash}(T^*, \text{ADM})$ Type 2b Forgery: $T \in \text{span}_{\vdash}(T^*, \text{ADM})$ To win $\mathcal{A}_{unf}$, the attacker must be able to construct one of the three above forgeries. This forgery can be used to break at least one of the underlying primitives.

Type 1 Forgery: In the first case, we can use the Type 1 Forgery of $\mathcal{A}_{unf}$ to create $\mathcal{A}_{unfSig}$ which forges a signature. We construct $\mathcal{A}_{unfSig}$ using $\mathcal{A}_{unf}$ as follows:
(1) $\mathcal{A}_{unfSig}$ chooses a hash-function $\mathcal{AH}$ and passes *prm* to $\mathcal{A}_{unf}$. This is also true for *pk* of the signature scheme to forge.
(2) All queries to O^{Sign} from $\mathcal{A}_{unf}$ are forwarded to $\mathcal{A}_{unfSig}$'s oracle and genuinely returned to $\mathcal{A}_{unf}$.
(3) Eventually, $\mathcal{A}_{unf}$ will output a pair (T^*, σ_T^*). $\mathcal{A}_{unfSig}$ returns (T^*, σ_T^*), as a valid forgery. The concrete signature forged can easily be extracted by defining a tree-traversal algorithm looking for the signature not queried for the particular value. This is due to the fact, that we allow to distribute sub-trees. Hence, any node may be forged, not just the root node.

Type 2a Forgery: In the case of 2a, we can use the Type 2a Forgery produced by $\mathcal{A}_{unf}$ to construct $\mathcal{A}_{col}$ which breaks the collision-resistance of the underlying hash-function. To do so, (1) $\mathcal{A}_{col}$ generates a key pair of a signature scheme to emulate O^{Sign} and chooses $\mathcal{AH}$.
(2) It passes *pk* and *prm* to $\mathcal{A}_{unf}$.
(3) For every request to the signing oracle, $\mathcal{A}_{col}$ generates the signature σ using *sk* and returns it to $\mathcal{A}_{unf}$.
(4) Eventually, $\mathcal{A}_{col}$ will output (T^*, σ_T^*). Given the transcript of the simulation, $\mathcal{A}_{col}$ searches for a pair $\mathcal{MH}(n_1) = \mathcal{MH}(n_2)$ with different content resp. sub-trees. If such a pair is found and $T^* \notin \text{span}_{\vdash}(T_i, \text{ADM}_i)$, $\mathcal{A}_{col}$ outputs exactly this pair, else it aborts. The outputted pair is a collision of the hash-function.

Type 2b Forgery: If $\mathcal{A}_{unf}$ returns a Type 2b Forgery, we can build $\mathcal{A}_{one}$ which calculates membership proofs of the underlying accumulator. To do so, (1) $\mathcal{A}_{one}$ generates a key pair of a signature scheme to emulate O^{Sign} and chooses $\mathcal{AH}$.
(2) It passes *pk* and $\mathcal{AH}$ to $\mathcal{A}_{unf}$.
(3) For every request to the signing oracle, $\mathcal{A}_{one}$ generates the signature σ using *sk* and returns it to $\mathcal{A}_{unf}$.
(4) Eventually, $\mathcal{A}_{unf}$ will output (T^*, σ_T^*). Given the transcript of the simulation, $\mathcal{A}_{one}$ searches for a pair $\mathcal{MH}(n_1) = \mathcal{MH}(n_2)$ with different content resp. sub-trees. If such a pair is found and $T_i \in \text{span}_{\vdash}(T^*, \text{ADM}^*)$, $\mathcal{A}_{one}$ outputs $(T_i, T^*, \sigma_T, \sigma^*)$, iff the preimage maps to queried document. In other words, the queried tree must be in the transitive closure of the preimage. Otherwise, we just have a normal collision, which belongs to case 2a. The membership proofs of the used accumulator can trivially be extracted. We showed how to use all three forgery types to break existential unforgeability of the underlying signature scheme Π, the one-way or the collision-resistance property of $\mathcal{AH}$. $\square$

Our construction is transparent and private. If $\mathcal{AH}$ always outputs uniformly distributed digests, and the digests and witnesses are therefore indistinguishable from random numbers, our scheme is transparent and therefore also private (Brzuska et al., 2010a): This follows directly from the definitions, i.e., the uniform distribution of the digests and witnesses from random numbers. In particular, all output of $\mathcal{AH}$ is computationally indistinguishable from random. This implies that the output of $O^{ModifyOrSign}$ is also computationally indistinguishable from uniform, hence hiding the secret bit b with overwhelming probability. In other words, an adversary breaking transparency is able to distinguish between random and computed digests, which has been assumed to be in-

feasible. Attacking the nonces is not possible, since removing a random from a uniform distribution results in a uniform distribution again. An additional note: This is the reason why we require a random seed for the accumulator; otherwise, an adversary could just recalculate the digests. □

B Formal Definition of the Requirements of $\mathcal{AH}$

Collision-Resistance and One-Wayness. The family $\mathcal{AH}_K$ contains only collision-resistant functions (Barić and Pfitzmann, 1997). Furthermore, it must be hard to find a digest with the same value without having the preimages, i.e., we need strong one-wayness (Barić and Pfitzmann, 1997). We can capture both requirements:

$$\Pr[k \xleftarrow{\$} K; x \xleftarrow{\$} \mathcal{X}_k; y \xleftarrow{\$} \mathcal{Y}_k; (x', y') \leftarrow \mathcal{A}(x, y) : \mathcal{AH}_k(x, y) = \mathcal{AH}_k(x', y') \wedge y \neq y'] < \varepsilon(\lambda)$$

Where the probability is taken over all coin tosses. In other words, an adversary should not be able to reverse the hashing step and to find a valid preimage or to find any other collision.

Indistinguishability of Output. We require that an adversary cannot decide how many additional members have been digested. We say that an accumulator is indistinguishable, iff the probability that the game depicted in Fig. B returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ).

Experiment $\mathsf{Indistinguishable}_{\mathcal{A}}^{\mathcal{AH}}(\lambda)$

$pk \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{parm})$
$b \xleftarrow{\$} \{0,1\}$
$d^* \leftarrow \mathcal{A}^{\mathsf{LoRHash}(\ldots,b,pk)}(\omega, \mathsf{parm}, pk)$
where oracle $\mathsf{LoRHash}$ for input $\mathcal{S}, \mathcal{R}$:
$z \xleftarrow{\$} \mathcal{Y}_{pk}$
if $b = 1$: return $(\mathsf{Hash}(pk, \mathcal{S} \cup \mathcal{R} \cup z),$
$\{(y_i, p_i) \mid p_i \leftarrow \mathsf{Proof}(pk, y_i \in \mathcal{S}, \mathcal{S} \cup \mathcal{R} \cup z)\})$
if $b = 0$: return $(\mathsf{Hash}(pk, \mathcal{S} \cup z),$
$\{(y_i, p_i) \mid p_i \leftarrow \mathsf{Proof}(pk, y_i \in \mathcal{S}, \mathcal{S} \cup z)\})$
return 1, iff $d = b$

Here, the adversary can choose the input, and has to guess, if the digest has only one more member ($b = 0$) or more ($b = 1$). The blinding value z is chosen by the oracle at random. The basic idea is to require that one additionally accumulated blinding value is enough to hide that an accumulator has additional members. Please note that the witnesses are also returned.

C *Brzuska* et al.'s Security Model

In (Brzuska et al., 2010a) *Brzuska* et al. formalized the needs of signatures for tree-structured documents. A RSS for tree-structured documents requires four efficient algorithms; in particular $\mathcal{RSS}_T := (\mathsf{sKg}, \mathsf{sSign}, \mathsf{sVf}, \mathsf{sCut})$:

- $\mathsf{sKg}(1^\lambda$ outputs the key-pair (sk, pk), where λ is the security parameter;
- $\mathsf{sSign}(sk, T)$ outputs a structural signature σ_T;
- $\mathsf{sVf}(pk, T, \sigma_T)$ outputs a bit $v \in \{0,1\}$, which indicates the correctness of the signature σ_T protecting the tree T and
- the redaction algorithm $\mathsf{sCut}(pk, T, \sigma_T, L_i)$, which removes the leaf L_i from the tree T and outputs a sub-tree $T' \simeq T \setminus \{L_i\} \preceq T$ and the corresponding new signature σ'_T for which $\mathsf{sVf}(pk, T', \sigma'_T)$ outputs 1.

Applying the leaf-cutting algorithm `sCut` subsequently allows removing complete sub-trees (Brzuska et al., 2010a). It does not allow to redact non-leaves or to express re-locations of sub-trees.

RSS Security Requirements We informally repeat the existing security properties for tree-structured documents as given and formalized by *Brzuska* et al. in (Brzuska et al., 2010a). These requirements should also hold for the structure of the tree T, not just its data. The structural integrity protection requires that all relations between nodes and their position within the tree's hierarchy are protected by the signature σ_T.

1. *Unforgeability:* No one should be able to compute a valid signature on a tree T' for pk without having access to the corresponding secret key sk. This is analogous to the standard unforgeability requirement for signature schemes, as already noted in (Brzuska et al., 2010a).
2. *Privacy:* Given a sub-tree with a signature σ and two possible source trees $T_{j,0}$ and $T_{j,1}$, no one should be able to decide from which source tree the , stems from. This definition is similar to the standard indistinguishability notation for encryption schemes (Brzuska et al., 2010a).
3. *Transparency:* A third party should not be able to decide which operations may have been performed on a signed tree. Hence, whether a signature σ_T of a tree has been created from scratch

Experiment $\mathsf{Unforgeability}^{\mathsf{RSS}}_{\mathcal{A}}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(T^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{DSign}(sk,\cdot)}(pk)$
let $i = 1, 2, \ldots, q$ index the queries
return 1 iff
$\mathsf{DVerify}(pk, T^*, \sigma^*) = 1$ and
for all $1 \leq i \leq q$, $T^* \not\preceq T_i$

Figure 12: Game for Unforgeability

Experiment $\mathsf{Transparency}^{\mathsf{RSS}}_{\mathcal{A}}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$d \leftarrow \mathcal{A}^{\mathsf{DSign}(sk,\cdot),\mathsf{DSign/DCut}(\cdot,\cdot,sk,b)}(pk)$
where oracle $\mathsf{DSign/DCut}$ for input T, L:
if L is not a leaf of T, return $\perp$
if $b = 0$: $(T, \sigma) \leftarrow \mathsf{DSign}(sk, T)$,
$(T', \sigma') \leftarrow \mathsf{DCut}(pk, T, \sigma, L)$
if $b = 1$: $T' \leftarrow T \setminus L$
$(T', \sigma') \leftarrow \mathsf{DSign}(sk, T')$,
finally return (T', σ').
return 1 iff $b = d$

Figure 13: Game for Transparency

or through sCut shall remain indistinguishable for a party receiving a signed tree T (Brzuska et al., 2010a).

The given notations take the tree structure of documents into account and allow public redactions, as sCut only requires the public key pk. The formal games are depicted in Fig. 12, Fig. 13 and Fig. 14/15.

D Revisiting the KB-Scheme

Here we shortly review the KB-Scheme as introduced in (Kundu and Bertino, 2009). We omit the step that randomizes the traversal numbers preserving their ordering. The KB-Scheme claims this is done to preserve transparency. It has been shown in (Brzuska et al., 2010a) that this is not sufficient to maintain transparency. We already state the scheme in our notation, since the original notation in (Kundu and Bertino, 2009) and in (Brzuska et al., 2010a) is

Experiment $\mathsf{Privacy}^{\mathsf{RSS}}_{\mathcal{A}}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$d \leftarrow \mathcal{A}^{\mathsf{DSign}(sk,\cdot),\mathsf{SignCut}(\ldots,sk,b)}(pk)$
return 1 iff $b = d$

Figure 14: Game for Privacy

$\mathsf{SignCut}(T_{j,0}, L_{j,0}, T_{j,1}, L_{j,1}, sk, b)$
if $T_{j,0} \setminus L_{j,0} \not\cong T_{j,1} \setminus L_{j,1}$ return $\perp$
$(T_{j,b}, \sigma_{j,b}) \leftarrow \mathsf{sSign}(sk, T_{j,b})$
return $(T'_{j,b}, \sigma'_{j,b}) \leftarrow \mathsf{sCut}(pk, T_{j,b}, \sigma_{j,b}, L_{j,b})$

Figure 15: SignCut Oracle

not able to express the possibility of removing intermediate nodes. Every node $n_i \in T$ will be addressed by his pre-order traversal number, i.e., the root is denoted as n_1. Note, the amount of nodes in T, i.e., $|V|$, will be denoted as n.

KeyGen. Generate a key pair (sk, pk) of an aggregating signature scheme $\mathcal{ASS}$ allowing public aggregation, e.g., the BGLS-Scheme (Boneh et al., 2003). By abuse of notation, we assume that pk contains all system parameters.

Sign. The signing-step outputs a signature for each node inside the tree:

1. Compute the pre- and post-order traversal numbers, of the tree T.
2. Transform these lists into an randomized but order-preserving space. For each node n_i, let ρ_i denote the associated pair of randomized traversal numbers
3. Set $G_T \leftarrow \mathcal{H}(\omega||\rho_1||c_1||\ldots||\rho_n||c_n)$, where ω is a nonce and $\mathcal{H}$ a cryptographic hash-function like SHA-512
4. $\forall n_i \in T$ compute: $\xi_i \leftarrow \mathcal{H}(G_T||\rho_i||c_i)$
5. Sign all ξ_i, i.e., $\sigma_i \leftarrow SIGN_{\mathcal{ASS}}(sk, \xi_i)$
6. Aggregate all signatures into σ_T
7. Output $\sigma = (T, \sigma_T, \{(\sigma_i, \rho_i)\}_{0<i\leq n}, G_T, pk)$

Modify. The *Kundu*-Scheme allows redaction of arbitrary nodes but no re-locations. Hence, MOD just contains the description to redact the node n_i:

1. Verify the signature using Verify
2. Remove n_i from T, i.e., $T' \leftarrow \text{MOD}(T)$. This can be expressed as $T' \leftarrow T \setminus n_i$, where n_i is the node to be redacted as specified by MOD. Both also includes all edges from resp. to n_i. Note, n_i may not be a leaf
3. Aggregate all signatures $\{\sigma_j\}_{j\neq i}$ into σ'_T
4. Output the altered tuple σ', i.e.,:

$$\sigma' = (T', \sigma'_T, \{(\sigma_i, \rho_i)\}_{0<i\leq n'}, G_T, pk)$$

Verify. Verification just uses σ:

1. For each node $n_i \in T$ compute $\xi_i \leftarrow \mathcal{H}(G_T||\rho_i||c_i)$
2. Check the validity using $\mathcal{ASS}$. In particular, each ξ_i calculated must be signed and contained in σ_T

3. Traverse the tree using pre-order and check if each of the associated traversal numbers is in the correct order, i.e., the associated pre- and post-order must remain plausible. Let f denote the parent of g; it must yield that $p_f > p_g$ and $r_f < r_g$, where p_x denotes the associated pre-order-number and r_x the post-order-number associated to the node x.
4. Output 1, if all checks pass, 0 otherwise resp. $\perp$ on error