On Structural Signatures for Tree Data Structures

Kai Samelin[1,⋆], Henrich C. Pöhls[2,⋆⋆], Arne Bilzhause[2], Joachim Posegga[2], and Hermann de Meer[1]

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

Abstract. In this paper, we present new attacks on the redactable signature scheme introduced by *Kundu* and *Bertino* at VLDB '08. This extends the work done by *Brzuska* et al. at ACNS '10 and *Samelin* et al. at ISPEC '12. The attacks address unforgeability, transparency and privacy. Based on the ideas of *Kundu* and *Bertino*, we introduce a new provably secure construction. The corresponding security model is more flexible than the one introduced by *Brzuska* et al. Moreover, we have implemented the schemes introduced by *Brzuska* et al. and *Kundu* and *Bertino*. The practical evaluation shows that schemes with a quadratic complexity become unuseable very fast.

1 Introduction

A redactable signature scheme (RSS) allows a third party, we name *sanitizer*, to redact contents of a signed document $m = m[1]||\dots||m[n]$ to generate a sanitized version of the document, which signature is still valid. This action can be performed without involvement of the original signer and without knowing any private keys. In more detail, a RSS allows to replace a block $m[i] \in m$ with $\emptyset$. Thus, a redaction leaves a blinded document m', where $m' = m \setminus m[i]$. Still, a third party is able to verify that all received blocks and their ordering are authentic. *Kundu* and *Bertino* were the first to apply this paradigm to tree-structured data [12]. In this paper, we present new attacks on the scheme introduced by *Kundu* and *Bertino* [12]. This extends the work done by *Brzuska* et al. [5] and *Samelin* et al. [20]. We introduce a provably secure scheme based on *Kundu* and *Bertino*'s initial idea. We have implemented the first version of *Kundu* and *Bertino*'s scheme [12] and the secure scheme by *Brzuska* et al. [5] and provide a detailed performance analysis. Our construction's performance is comparable to the one introduced in [12] by *Kundu* and *Bertino*: the runtime and storage complexity is in $\mathcal{O}(n)$, if one uses a RSS for lists, that is also in $\mathcal{O}(n)$ for both

⋆ Is supported by "Regionale Wettbewerbsfähigkeit und Beschäftigung", Bayern, 2007-2013 (EFRE) as part of the SECBIT project. (http://www.secbit.de)
⋆⋆ Is funded by BMBF (FKZ:13N10966) and ANR as part of the ReSCUeIT project.

F. Bao, P. Samarati, and J. Zhou (Eds.): ACNS 2012, LNCS 7341, pp. 171–187, 2012.

metrics. The used RSS must protect the order among its elements. As an example, the scheme introduced in [19] achieves this. Other existing provably secure schemes for trees have a runtime and storage complexity of $\mathcal{O}(n^2)$ [5].

State of the Art. The term RSS was coined in 2002 by *Johnson* et al. in [10]. In the same year, *Steinfeld* and *Bull* introduced the same concept as "Content Extraction Signatures" [21]. Since then, RSSs have been extended to tree-structured data [5,12,13] and to arbitrary graphs [15]. *Samelin* et al. introduced the concept of independently redactable structure in [20].

A related concept are Sanitizable Signature Schemes (SSS), introduced by *Ateniese* et al. in [3]. In a SSS, the sanitizer does not delete blocks, but can modify $m[i]$ into an arbitrary string $m[i]' \in \{0,1\}^*$ [6]. It is possible to restrict sanitizers in SSSs to certain values. This is a well-known field; refer to [7,11] for related work. Approaches to merge SSSs and tree-structured data have also been published, e.g., in [18]. *Pöhls* et al. show in [18] that RSSs for lists are not suitable for tree-structured documents in certain scenarios. We only discuss RSSs in this paper.

Brzuska et al. defined and formalized a set of desired properties for redactable tree-structured documents in [5]. *Kundu* and *Bertino* showed in [13], that non-private RSS can be attacked using "side-channel" information, i.e., if the position of a redacted block is visible, one may be able to reconstruct some information. They name these type of vulnerability "inference attacks" [13]. Consider the following example clarifying this statement: in a tree-based patient record of a hospital, which has exactly two wards, i.e., cancer and surgery, a sub-tree represents the treatments in each ward. If a patient has been treated in both wards, and one subtree has been removed, an adversary sees $\emptyset$. It can deduce that the patient has been treated in both wards, using the knowledge that two wards exist and the information about the structure of the patient record. This impacts on privacy and is not acceptable in certain scenarios. The scheme introduced in [12] by *Kundu* and *Bertino* was originally proposed to address these problems. However, two attacks on this scheme have been published already: one attacking transparency and privacy [5] and one attacking structural integrity [20].

Our Contribution. The scheme introduced in [12] by *Kundu* and *Bertino* has some very useful properties. In particular, it has a low runtime and storage complexity, i.e., both are $\mathcal{O}(n)$, where n is the number of vertices in a tree $T = (V, E)$. This makes their scheme the fastest one known to the authors for both metrics. Other provably secure and transparent schemes proposed, e.g., [5], have a complexity of $\mathcal{O}(n^2)$, are only useable for lists [8,20], or are only able to quote substrings [2]. However, the scheme introduced in [2] fulfills a very strong privacy notation, i.e., strong context-hiding. This notation prohibits even the signer from deciding if two quotes have been derived from the same source. Our goal is a secure scheme not limited to quoting for trees. The worst-case approximation of the scheme introduced by *Brzuska* et al. in [5] depends on the branching factor, which in the case of a tree with height 1, is $\mathcal{O}(n)$, where n is the number of leafs. As the branching factor is not a constant factor anymore, the growth will

be quadratic in n. All other current schemes in $\mathcal{O}(n)$, e.g., the ones introduced in [9] or [13], are susceptible to the attack on privacy introduced by *Brzuska* et al.'s for the scheme by *Kundu* and *Bertino* in [5]. This is due to fact that appending *ordered* random numbers cannot be sufficient to hide redactions [5]. We use the ideas of *Kundu* and *Bertino* [12,13] to derive a new construction which is provably secure and does not inherit their flaws. We contribute by adding new attacks, breaking unforgeability, transparency and privacy. We derive a provably secure construction based on their ideas. Moreover, we have implemented the schemes by *Brzuska* et al. [5] and the scheme introduced in [12] and show that schemes with a quadratic runtime complexity become unuseable very fast.

Outline of the Paper. The rest of the paper is structured as follows. In Sect. 2, we extend the existing definitions required to understand the schemes presented. We will shortly restate the scheme by *Kundu* and *Bertino* [12], along with the new attack vectors. Our new scheme is presented in Sect. 3. We extend the new scheme to add additional useful properties, as proposed in [17] and [20], in Sect. 4. We present our implementations along with the corresponding performance analysis in Sect. 5. Finally, Sect. 6 concludes our work. All formal proofs can be found in the appendices.

2 Preliminaries, the Scheme by *Kundu* and *Bertino* and the Extended Security Model

We start this section by defining the algorithms of an RSS in general. Our notation is inspired by *Brzuska* et al. [5], but extended to allow redaction of non-leafs. The redaction of non-leafs is allowed in the schemes by *Kundu* and *Bertino* and offers more flexibility [12,13]. However, securely allowing this flexibility requires a more sophistacted approach. A thorough discussion is given in Sect. 2.3.

Definition 1 (Redactable Signature Scheme). *A RSS for trees $\mathcal{RSS}_T$ consists of four PPT algorithms: $\mathcal{RSS}_T :=$ (KeyGen, TSign, TVerify, TShare).*

KeyGen. *The key generation algorithm* $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *outputs a key pair consisting of the private key* sk *and the public key* pk*:* $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, on input of a security parameter* λ*.*

TSign. *The signing algorithm* $\mathsf{TSign}(\mathrm{sk}, T, r, i)$ *takes as input the secret key* sk*, the tree* T*, a flag* $r \in \{0,1\}$ *indicating, if the root is allowed to be redacted, and a flag* $i = \{0,1\}$ *which indicates, if non-leafs can be redacted. We define that a* 1 *indicates that the corresponding action is allowed. It outputs a signature* σ_T *over the tree* T*:* $(T, \sigma_T) \leftarrow \mathsf{TSign}(\mathrm{sk}, T, r, i)$

TVerify. *The algorithm* $\mathsf{TVerify}(T, \mathrm{pk}, \sigma_T)$ *takes as input a tree* T*, a public key* pk *and the signature* σ_T*. It outputs a bit* $d \in \{0,1\}$*, which indicates, if* σ_T *is a valid* $(d = 1)$ *signature on* T *under the public key* pk*:* $d \leftarrow \mathsf{TVerify}(T, \mathrm{pk}, \sigma_T)$

TShare. *The algorithm* $\mathsf{TShare}(T, pk, \sigma_T, \mathcal{N})$ *takes as input a tree* T*, a public key* pk *and signature* σ_T*, as well as a set of nodes* $\mathcal{N} \subseteq T$ *to redact. It returns a new tree* $T' \leftarrow \mathrm{MOD}(T, \mathcal{N})$*, along with a new signature* σ'_T *resp.* $\perp$ *on error, where* MOD *is the function modifying the tree* T *w.r.t.* $\mathcal{N}$*, i.e.,*

$T' = T \setminus \mathcal{N}$. Hence, TShare outputs $(T', \sigma'_T) \leftarrow \mathsf{TShare}(T, \mathrm{pk}, \sigma_T, \mathcal{N})$, resp. $\perp$ on error. We do not consider, if and how the edges between the nodes are treated, if a redaction takes place. This is the concern of an instantiation of a $\mathcal{RSS}_T$. In particular, it depends on the concrete use case, if non-leafs are allowed to be redacted. We want to leave this choice to the signer.

All algorithms must fulfill the usual correctness requirements. In particular, all genuinely signed trees and trees created from them by TShare must verify. This also implies that TShare always outputs a valid tree. How to actually ensure this and what a valid tree in terms of the signature is, depends on the algorithms and use cases. Note, if any result of the algorithms given is not a tree, the result is treated as $\perp$. We state the extended security model next.

The Extended Security Model. *Brzuska* et al. introduced and formalized some security properties in [5]. Their model was the first rigid approach. It is restrictive by allowing the cutting of leafs of the tree only. We extend their model, since the extended functionality of non-leaf redaction requires an adjustment. We denote the transitive closure of T, w.r.t. TShare, as $span_{\vdash}(T)$, following [8] and [20]. $b \xleftarrow{\$} \{0,1\}$ denotes that b is a random bit, drawn from a uniform distribution. The following security properties have been derived from [5]. As in the original model, the following definitions cater only for the information an adversary can derive from the signature. If obvious redactions took place, that are detectable or reversible using side-channel information, it may be trivial to decide whether something has been redacted.

1. *Unforgeability:* No one should be able to compute a valid signature on a tree T^* verifying under pk outside $span_{\vdash}(T)$, without access to the corresponding secret key sk. This is analogous to the standard unforgeability requirement for signature schemes, as already noted in [5]. A scheme $\mathcal{RSS}_T$ is unforgeable, iff for any efficient (PPT) adversary $\mathcal{A}$, the probability that the game depicted in Fig. 1 returns 1, is negligible (as a function of λ). In this game, the adversary has access to a signing oracle.
2. *Privacy:* No one should be able to gain any knowledge about the unmodified tree from a redacted version without having access to the original. This is similar to the standard indistinguishability notation for encryption schemes [5]. We say that a scheme $\mathcal{RSS}_T$ is private, iff for any efficient (PPT) adversary $\mathcal{A}$, the probability that the game shown in Fig. 3 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ). In this game, the adversary has to figure out, which input has been used by the LoR-Oracle.
3. *Transparency:* A third party should not be able to decide whether a signature σ_T of a tree T has been created from scratch or through TShare. In other words, a party who receives a signed tree T cannot tell whether he received a freshly signed tree or a tree which has potentially been modified [5]. We say that a scheme $\mathcal{RSS}_T$ is transparent, iff for any efficient (PPT) adversary $\mathcal{A}$, the probability that the game shown in Fig. 2 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ). In this game, the adversary has to figure out, if

Experiment $\mathsf{Unforgeability}^{\mathsf{RSS}_T}_{\mathcal{A}}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(T^*, \sigma^*_T) \leftarrow \mathcal{A}^{\mathsf{TSign}(sk,\dots)}(pk)$
let $i = 1, 2, \dots, q$ index the queries to the signing oracle
return 1 iff
$\mathsf{TVerify}(T^*, pk, \sigma^*_T) = 1$ and
$\forall i : 0 < i \leq q,\ T^* \notin span_{\vdash}(T_i)$

Fig. 1. Game for Unforgeability

Experiment $\mathsf{Transparency}^{\mathsf{RSS}_T}_{\mathcal{A}}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$b \xleftarrow{\$} \{0, 1\}$
$d \leftarrow \mathcal{A}^{\mathsf{TSign}(sk,\cdot),\mathsf{Adapt/Sign}(\dots,sk,b)}(pk)$
where oracle Adapt/Sign for input $T, \mathcal{N}$:
if $\mathcal{N} \not\subseteq T$, return $\perp$
if $b = 0$: $(T, \sigma_T) \leftarrow \mathsf{TSign}(sk, T, r, i)$,
$(T', \sigma'_T) \leftarrow \mathsf{TShare}(T, pk, \sigma_T, \mathcal{N})$
if $b = 1$: $T' \leftarrow \textsc{mod}(T, \mathcal{N})$
$(T', \sigma'_T) \leftarrow \mathsf{TSign}(sk, T', r, i)$,
return (T', σ'_T).
return 1 iff $b = d$

Fig. 2. Game for Transparency

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

Fig. 3. Game for Privacy

$\mathsf{LoRAdapt}(T_{j,0}, \mathcal{N}_{j,0}, T_{j,1}, \mathcal{N}_{j,1}, r, i, sk, b)$
if $\textsc{mod}(T_{j,0}, \mathcal{N}_{j,0}) \neq \textsc{mod}(T_{j,1}, \mathcal{N}_{j,1})$ return $\perp$
$(T_j, \sigma_{T_j}) \leftarrow \mathsf{TSign}(sk, T_{j,b}, r, i)$
$(T'_j, \sigma'_{T_j}) \leftarrow \mathsf{TShare}(T_{j,b}, pk, \sigma_{T_{j,b}}, \mathcal{N}_{j,b})$
return $(T'_{j,b}, \sigma'_{T,b})$

Fig. 4. LoRAdapt Oracle for Privacy

the signature has been created through TSign or TShare. This encapsulated by the Adapt/Sign-Oracle.

Implications and Seperations. The implications given by *Brzuska* et al. in [5] and [6] do not change: transparency $\implies$ privacy, privacy $\not\Rightarrow$ transparency, and unforgeability is independent. To avoid duplicate work, we omit the proofs, as only minor adjustments are required.

2.1 Aggregate Signatures and Bilinear Pairings

Aggregate signatures ($\mathcal{AGG}$) have been introduced by *Boneh* et al. in [4]. The basic idea is as follows: given ℓ signatures, i.e., $\{\sigma_i \mid 0 < i \leq \ell\}$, a $\mathcal{AGG}$ constructs one compressed signature σ_c which contains all signatures σ_i. This allows verifying all given signatures σ_i by verifying σ_c. To construct such a scheme, let $\mathbb{G}_1$ be a cyclic multiplicative group with prime order q, generated by g, i.e., $\mathbb{G}_1 = \langle g \rangle$. Further, let $\mathbb{G}_T$ denote a cyclic multiplicative group with the same prime order q. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, where $\mathbb{G}_T = \langle \hat{e}(g, g) \rangle$, be a bilinear map such that:

1. Bilinearity: $\forall u, v \in \mathbb{G}_1 : \forall a, b \in \mathbb{Z}/q\mathbb{Z} : \hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$
2. Non-degeneracy: $\exists u, v \in \mathbb{G}_1 : \hat{e}(u, v) \neq 1$
3. Computability: There is an efficient algorithm $\mathcal{A}_{bimap}$ that calculates the mapping $\hat{e}$ for all $u, v \in \mathbb{G}_1$

Definition 2 (The BGLS-Scheme). *The $\mathcal{AGG}$ by* Boneh *et al. [4] (BGLS-Scheme) with public aggregation consists of five efficient algorithms. We will only use one public key, Q, which allows a performance improvement, while making sure that just one signing key is used. Next, we define:*

$$\mathcal{AGG} := (\mathit{AKeyGen}, \mathit{ASign}, \mathit{AVerf}, \mathit{AAgg}, \mathit{AAggVerf})$$

AKeyGen. *The algorithm* KeyGen *outputs the public and private key of the signer, i.e.,* $(\mathrm{pk}, \mathrm{sk})$. $\mathrm{sk} \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ *denote the signer's private key. Additionally, let* $\mathcal{H}_k : \{0,1\}^* \to \mathbb{G}_1$ *be an ordinary cryptographic hash-function from the family* $\mathcal{H}_K$*, modeled as a random oracle, and set* $Q \leftarrow g^{\mathrm{sk}}$*, where* $\langle g \rangle = \mathbb{G}_1$*. Set the public parameters and key* $\mathrm{pk} \leftarrow (g, Q, \mathbb{G}_1, \mathbb{G}_T, \mathcal{H}_k, \hat{e})$.

ASign. *The algorithm* ASign *outputs the signature* σ_i *on input of the secret key* sk *and a single string* $m_i \in \{0,1\}^*$*:* $\sigma_i \leftarrow (\mathcal{H}_k(m_i))^{\mathrm{sk}}$.

AVerf. *To verify a signature* σ_i*, check, if the following equation holds:* $\hat{e}(\sigma_i, g) \stackrel{?}{=} \hat{e}(\mathcal{H}_k(m_i), Q)$.

AAgg. *To aggregate* ℓ *signatures* σ_i *into an aggregated signature* σ_c*, the aggregator computes* $\sigma_c \leftarrow \prod_{i=1}^{\ell} \sigma_i$*, denoted as* $\mathsf{AAgg}(\mathrm{pk}, \mathcal{S})$*, where* $\mathcal{S}$ *is the set of* ℓ *signatures signed using the same public parameters contained in* pk. AAgg *can be used by untrusted parties and without knowing the private key.*

AAggVerf. *To verify an aggregated signature* σ_c*, check whether* $\hat{e}(\sigma, g) \stackrel{?}{=} \prod_{i=1}^{\ell} \hat{e}(\mathcal{H}_k(m_i), Q)$ *holds, on input of* σ_c*,* pk *and a list of all signed* m_i*. To improve efficiency, the right side can be rewritten as* $\hat{e}(\prod_{i=1}^{\ell} \mathcal{H}_k(m_i), Q)$*, due to the use of only one public key. We denote the algorithm as* $d \leftarrow \mathsf{AAggVerf}(\mathrm{pk}, \sigma, \{m_i\}_{0<i\leq\ell})$.

The usual correctness requirements must hold, which have been formally proven in [4]. Moreover, we require the expected security properties to hold, i.e., unforgeability under chosen message attacks (UNF-CMA) and the k-element extraction assumption. The proofs and formal definitions can also be found in [4].

2.2 The Scheme by *Kundu* and *Bertino*

In this section, we shortly restate the scheme by *Kundu* and *Bertino* [12]. Afterwards, we describe the attacks. We use the following notations: n denotes the number of nodes (i.e., $|V|$); a node i is denoted as n_i; c_i denotes the label or content of n_i. A family of cryptographic hash-functions is denoted as $\mathcal{H}_{\mathcal{S}}$, where $\mathcal{S}$ denotes the key space of the hash function family. The following algorithm is the original one introduced in [12]. In their scheme, the input flags r and i are both fixed to 1. This indicates, that both, root and non-leaf node redaction, is always allowed. If nodes have to be ordered, we assume that the ordering algorithm used is known to every party involved, e.g., pre-order traversal. "||" denotes a concatenation which is uniquely reversible.

Construction 1 (The Kundu-Scheme.) *The scheme by Kundu and Bertino* $\mathcal{KS}$ *consists of four efficient algorithms. In particular* $\mathcal{KS} := (\mathsf{KeyGen}, \mathsf{TSign}, \mathsf{TVerify}, \mathsf{TShare})$.

KeyGen. *The key generation algorithm* $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *outputs a key pair of an aggregate signature scheme. It outputs* $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathcal{AGG}.\mathsf{AKeyGen}(1^\lambda)$

TSign. *The signing algorithm* $(T, \sigma_T) \leftarrow \mathsf{TSign}(\mathrm{sk}, T, r, i)$ *takes the secret key* sk *and the tree* T*. The flags* r *and* i *are defined to be* 1*. To sign the tree, perform the following steps:*

1. *Choose a cryptographic hash-function* $\mathcal{H}_s \in \mathcal{H}_{\mathcal{S}}$
2. *Get the pre-order traversal numbers of each node* $n_i \in T$*, denoted* l_i
3. *Get the post-order traversal numbers of each node* $n_i \in T$*, denoted* r_i
4. *Apply the randomizing, but order-preserving function* θ *to both lists of traversal numbers, using some distribution* Δ*. The randomized counterparts is denoted as* l_i^r *resp.* r_i^r*. How to calculate this randomization step is not important; the required order-preserving behaviour is enough to undermine transparency and privacy (See Sect. 2.3)*
5. *Let* $\rho_i := (l_i^r, r_i^r)$*. Set* $G_T \leftarrow \mathcal{H}_s(\rho_1||c_1||\ldots||\rho_n||c_n)$
6. *For* $1 < i \leq n$*, compute* $\sigma_i \leftarrow \mathcal{AGG}.\mathsf{ASign}(\mathrm{sk}, \mathcal{H}_s(G_T||\rho_i||c_i))$
7. *Compress all resulting signatures* σ_i *into a compressed signature* σ_c*, i.e.,* $\sigma_c \leftarrow \mathcal{AGG}.\mathsf{AAgg}(\mathrm{pk}, \{\sigma_i\}_{0<i\leq n})$*.*
8. *Output* (T, σ_T)*, where* $\sigma_T = (\mathrm{pk}, \sigma_c, G_T, s, \{(\sigma_i, \rho_i)\}_{0<i\leq n})$

TVerify. *The algorithm* $\mathsf{TVerify}(T, \mathrm{pk}, \sigma_T)$ *takes as input the received tree* T*, a public key* pk *and the received signature* σ_T*. It outputs a bit* $d \in \{0, 1\}$*, which indicates, if the received* σ_T *is a valid and correct signature on* T *under the public key pk. It performs the following steps to do so:*

1. *Compute* $d \leftarrow \mathcal{AGG}.\mathsf{AAggVerf}(\mathrm{pk}, \sigma_c, \{\mathcal{H}_s(G_T||\rho_i||c_i)\}_{0<i\leq n})$*.*
   If $d = 0$*, output 0, else continue*
2. *Check, if each node is positioned correctly using both traversal numbers derived from* ρ_i*. If the positions are correct, output* 1*, else output* 0

TShare. *The algorithm* $\mathsf{TShare}(T, \mathrm{pk}, \sigma_T, \mathcal{N})$ *takes as input a tree* T*, a public key pk and the tree's signature* σ_T*, as well as a subset* $\mathcal{N} \subseteq T$ *of nodes*

1. *Check the validity of* σ_T *using* $\mathsf{TVerify}$
2. *Set* $T' \leftarrow T \setminus \mathcal{N}$*, and also remove the edge(s). If non-leaf nodes are subject to redaction, implicit edges are introduced: the redacted node is skipped in terms of the edge. See Sect. 2.4 for more details*
3. *Compute* $\sigma'_c \leftarrow \mathcal{AGG}.\mathsf{AAgg}(\mathrm{pk}, \{\sigma_i\}_{0<i\leq n'})$*, where* $n' = |V'|$*. Note, we have rearranged the indices to account for the redaction*
4. *Output* (T', σ'_T)*, where* $\sigma'_T = (\mathrm{pk}, \sigma'_c, G_T, s, \{(\sigma_i, \rho_i)\}_{0<i\leq n'})$

2.3 Attacks on *Kundu*'s Transparency and Privacy

Transparency and privacy can actually be attacked in many ways. We go through all known attacks. Some attacks can just be mounted on special revisions of the schemes by *Kundu* and *Bertino*. We state this for each attack. To have a complete list of attacks against the new scheme is shown to be resilient against, we restate all attacks known yet.

Randomized Traversal Numbers. *Kundu* and *Bertino* propose three ways to randomize the traversal numbers: [13]

1. **Sorted Random Numbers**: Generate $|V|$ random numbers and sort them.
2. **Order-Preserving Encryption**: Apply an order-preserving encryption scheme, e.g. [1], to the traversal numbers.
3. **Addition of Random Numbers**: Assign the numbers to the nodes by taking the previous traversal number and adding a random offset.

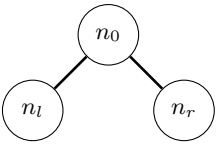

Fig. 5. Sample Tree I

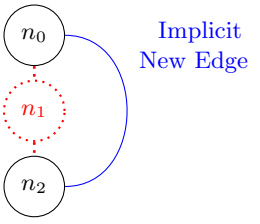

Fig. 6. Sample Tree II

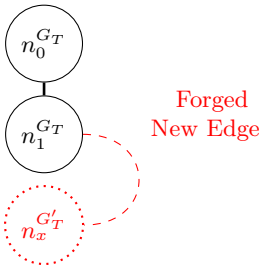

Fig. 7. Mix and Match

The following attack has been discovered by *Brzuska* et al. in [5]: Consider a tree with three nodes, i.e., the tree as depicted in Fig. 5. The algorithm θ outputs a tuple $(r_l, r_r, r_0) \leftarrow \theta(T, \Delta)$ on input of the depicted tree T in Fig. 5 and a distribution Δ. Assume that Δ is the uniform distribution and let $\mu = \mathrm{E}(r_l)$ be the expected random number associated to n_r. Furthermore, assume that $\Pr[r_l = \mu] = 0$ for a sufficiently large space. Hence, the probabilities are $\Pr[r_l < \mu] = \Pr[r_l > \mu] = 0.5$. Therefore, we obtain $\Pr[r_r > \mu] \geq 0.75$, since r_r is the largest random number and for $r_r < \mu$, both r_l and r_0 must be smaller than μ. Transparency can then be broken for the sample tree depicted in Fig. 5 as follows: transparency has been defined as a game where the adversary must guess a bit b. The game either returns a signature just containing n_0 and n_r or a signature which represents the whole tree where n_l has been cut off [5]. To win the game, the adversary just outputs 1, if $r_r \leq \mu$ and 0 otherwise. Note: $\Pr[r_r > \mu] \geq 0.75$ for a sanitized tree, while for a freshly signed one the probability changes: $\Pr[r_r > \mu] = 0.5$. Hence, b can be guessed with a non-negligible probability. Therefore, the scheme by *Kundu* and *Bertino* is neither transparent [5] nor private, since the leaked information allows to make statements about the original message. This attack works for other distributions as well, while it is required that the adversary knows the distribution Δ, which can be derived from the signing algorithm.

Leaking Structural Signature G_T. In the original paper [12], *Kundu* and *Bertino* introduce an additional digest which they name "structural signature", denoted as G_T. This "signature" is calculated as $G_T \leftarrow \mathcal{H}_s(\rho_1||c_1||\ldots||\rho_n||c_n)$. It is part of the calculation of each σ_i, i.e., $\sigma_i \leftarrow \mathcal{AGG}.\mathsf{ASign}(sk, G_T||\rho_i||c_i)$. Obviously, to check the signature σ_T, G_T needs to be available to the verifier. The verifier can calculate a digest G'_T on input of the tree T'. In particular, $G'_T \leftarrow \mathcal{H}_s(\rho_1||c_1||\ldots||\rho_{n'}||c_{n'})$, where $n' = |V'|$. Afterwards, the following equation is checked: $G'_T \stackrel{?}{=} G_T$. Following our definitions, this also destroys privacy. This problem has partially been circumvented in [13] by salting G_T; in particular the signer calculates: $G_T \leftarrow \mathcal{H}_s(\omega||\rho_1||c_1||\ldots||\rho_n||c_n)$, where ω is a nonce. However, this choice requires that $\mathcal{H}_s$ is modeled as a random oracle to ensure privacy [13]. In the revision introduced in [16], no structural signature is present, thus not inheriting this problem. However, omitting the tag renders that revision of the scheme forgeable, as we see next.

2.4 Attacks on *Kundu*'s Unforgeability and Structural Integrity

Besides the attacks on transparency and privacy, unforgeability can be attacked as well. In this section, two attacks are given. The first is adapted from [20]. The second attack is new.

Level Promotion. We shortly restate the observation which has recently been discovered by *Samelin* et al. in [20]. They show how to alter the semantic meaning of tree T by removing nodes which are not leafs. They name non-leafs "intermediate nodes". We adapt their nomenclature. Removing intermediate nodes allows introducing new implicit edges in certain scenarios. A sample tree, where this is possible, is depicted in Fig. 6. For this tree the traversal numbers are $(1, 2, 3)$ (pre-order) resp. $(3, 2, 1)$ (post-order). The randomization step can be omitted due to the order-preserving behaviour. If n_1 gets redacted, the traversal numbers of the remaining nodes are $(1, 3)$ (pre-order) resp $(3, 1)$, which, in terms of the signature, is valid. In particular, a sanitizer can introduce an edge $e_{n_0,n_2} \notin E$ which has not been signed by the signer. Strictly speaking, this destroys structural integrity, and therefore has a negative impact on unforgeability. Furthermore, this attack is possible in all versions of the schemes by *Kundu* and *Bertino* [12,13,16]. One might argue, that it depends on the definition of integrity, if this behavior can be considered as an attack. However, we think that such powerful possibilities must be under the sole control of the signer to avoid unwanted side-effects. This is in accordance with [20]. Our secured scheme allows such a level-promotion, but only after the signer explicitly allowed this behaviour during generation of the signature. Note, additionally the attack from [20] is also applicable, if one is able to redact parent nodes to allow distributing subtrees of a given tree. In our example, this would be the tree $T' = (\{n_1, n_2\}, \{e_{1,2}\})$. Obviously, this also leads to the same problem. This has not been mentioned in [20]. One may argue that intermediate node redaction lacks application scenarios. This is not true. Consider the following example: a tree's structure is implicitly describing the hierarchy within a company. Allowing to remove intermediate nodes now permits only to remove the department head. This leaves a list of employees. Without intermediate node redaction, this is not possible. Hence, there exist use cases, where removing leafs only is not sufficient.

Match-and-Mix Attacks. In the older revisions of the schemes by *Kundu* and *Bertino*, i.e., as proposed in [12,13], each node $n_x^{G_T}$ is bound to the tree T^{G_T} due to the structural signature G_T, which is part of each signature. Hence, a node $n_x^{G_T}$ cannot be merged into another tree $T^{G'_T}$, which nodes $n^{G'_T}$ have been signed with a different ***tag*** G'_T. However, in the newest revision [16], this tag is not present anymore. It is just aggregated onto the signature, but does not bind nodes to the tag, since it is not part of the signature generation of each node. In particular, $\sigma_i \leftarrow \mathcal{AGG}.\mathsf{ASign}(sk, G_T||\rho_i||c_i)$, where $G_T := \emptyset$. Hence, an adversary $\mathcal{A}$ can merge nodes of different trees into a new tree $T^{\mathcal{A}}$, which contains nodes originally signed for $T^{G'_T}$ and T^{G_T}. As an example, we show how the tree $T^{\mathcal{A}}$, as depicted in Fig. 7, can be constructed. W.l.o.g. let

$\rho_{n_0^{G_T}} = (1,3)$, $\rho_{n_1^{G_T}} = (2,2)$ be the nodes of tree T^{G_T} and $\rho_{n_x^{G'_T}} = (3,1)$ be the only node of $T^{G'_T}$. The adversary $\mathcal{A}$ has access to each of the individual signatures $\sigma_{n_0}, \sigma_{n_1}, \sigma_{n_x}$ and the "blinding signatures" σ_{ω_T} and $\sigma_{\omega'_T}$.[1] Hence, the adversary can construct a signature $\sigma_{\mathcal{A}}$ on the tree $T^{\mathcal{A}}$ by calculating $\sigma_{\mathcal{A}} \leftarrow \mathcal{AGG}.\mathsf{AAgg}(pk, \{\sigma_{n_0}, \sigma_{n_1}, \sigma_{n_x}, \sigma_{\omega_T}\})$ resp. $\sigma'_{\mathcal{A}} \leftarrow \mathcal{AGG}.\mathsf{AAgg}(pk, \{\sigma_{n_0}, \sigma_{n_1}, \sigma_{n_x}, \sigma_{\omega'_T}\})$, where $T^{\mathcal{A}}$ consists of all three nodes mentioned. Running $\mathsf{TVerify}(T^{\mathcal{A}}, pk, \sigma_{\mathcal{A}})$ outputs the bit 1. As a result, the attacker has successfully forged a signature for $T^{\mathcal{A}}$. This new attack can be applied as soon as two trees are signed with the same private key. Hence, the newest revision of the scheme by *Kundu* and *Bertino* is forgeable. Again, the older revisions do not have this problem, as every node $n_x^{G_T}$ is bound to the tree T^{G_T} due to G_T, which is unique for each tree signed. We conclude, that all revisions, i.e. [12,13,16], are not secure in our model.

3 Our New Secure Construction

We present a construction based on *Kundu* and *Bertino*'s ideas, but without inheriting the previously mentioned weaknesses. We use a transparent and unforgeable RSS for standard linear documents, which is denoted as Π in this section. In particular, let $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Redact}, \mathsf{Verify})$ be a secure RSS for lists. Moreover, we require transparency, privacy and unforgeability. Please refer to [5,8,20] for the formal definitions. The existing secure schemes have a runtime of $\mathcal{O}(n^2)$ or $\mathcal{O}(n \cdot log(n))$ resp. [2], whereas the latter just allows quoting of substrings, hence not suitable for arbitrary redactions. Their scheme fulfills a strong privacy notation, strong context-hiding. We are aware of the fact that *Kundu* et al. introduced a new scheme in [14]. However, their new scheme also has a worst case complexity of $\mathcal{O}(n^2)$, only allows to remove leafs and therefore offers **no** advantage to the one introduced in [5] by *Brzuska* et al. Additionally, [14] relies on a different idea, based on signing binary relations, similar to [5,8,20]. Hence, we see their scheme as a completely new construction not related to the original idea. Our scheme makes use of a RSS for lists. Utilizing a RSS in $\mathcal{O}(n)$, our new scheme is considerably faster than the existing ones. Such a RSS has been proposed by *Pöhls* et al. in [19].

Sketch of Our Secure Construction. We first sketch our construction to increase readability of the algorithmic descriptions. The basic idea is to use the fact that a tree is uniquely determined by its pre- and postorder traversal numbers. However, as *Brzuska* et al. show in [5], ordering random numbers leads to insecure schemes. Hence, we need to find a way to sign the ordering of the traversal numbers without explicitly ordering them. We achieve this by using a secure transparent RSS, which protects the order of the signed parts. We use this order preserving RSS to sign two lists. Each list contains $|V|$ uniformly distributed random nonces, which are pairwise unique. We use one list for pre- and another one for post-order traversal numbers. We annotate a node n_i with

[1] An adversary can always build an inverse of all signatures received.

the nonce at the positions in the lists corresponding to the node's traversal numbers. The ordering of each list secured by signing it with the RSS. Note, the nonces in the lists are not ordered. When a node n_i is redacted, the two nonces used to annotate n_i are removed from the respective list using the RSS, while leaving each list of remaining nonces still uniformly distributed. Additionally, the signer can decide, if a redaction of the root or redaction of intermediate nodes is allowed, as the signer annotates the nodes accordingly. This keeps the signer in control, while giving more flexibility. This must be done by annotating nodes accordingly. We cannot use the content of the nodes in the lists, as they may not be unique, making the reconstruction ambiguous and therefore forgeable.

Construction 2 (The New Flexible Construction.) *We give an algorithmic explanation of our scheme. The security proofs can be found in the appendices, but from the algorithmic description the new scheme's resilience against the mentioned attacks are visible. The aggregate signature is used to improve the performance of the verification process and to achieve consecutive redaction control, as we show in Sect. 4.1.*

KeyGen. *The key generation algorithm outputs two key pairs: (1) A key pair for an aggregate signature scheme $\mathcal{AGG}$, i.e., $(sk_{\mathcal{AGG}}, pk_{\mathcal{AGG}}) \leftarrow \mathcal{AGG}.\mathsf{AKeyGen}(1^\lambda)$. (2) A key pair for Π, i.e., $(sk_\Pi, pk_\Pi) \leftarrow \Pi.\mathsf{KeyGen}(1^\lambda)$. Output $(\mathrm{sk}, \mathrm{pk}) = ((\mathrm{sk}_\Pi, \mathrm{sk}_{\mathcal{AGG}}), (\mathrm{pk}_\Pi, \mathrm{pk}_{\mathcal{AGG}}))$*

TSign. *The signing algorithm $(T, \sigma_T) \leftarrow \mathsf{TSign}(\mathrm{sk}, T, r, i)$ takes all secret keys, the tree T to sign and the flags r and i. To sign T, perform:*

1. *Generate two lists, $\mathcal{L}$ and $\mathcal{M}$, each containing $n = |V|$ pairwise distinct uniformly distributed random numbers $\in \{0,1\}^\lambda$. The elements of the list are addressed by an index, i.e., $\mathcal{L}_i$*
2. *Let i be the index of a pre-order traversal $c_i \leftarrow \mathcal{L}_i || c_i$*
3. *Let j be the index of a post-order traveral. Set $c_i \leftarrow \mathcal{M}_j || c_i$*
4. *Choose a random tag $\tau \xleftarrow{\$} \{0,1\}^\lambda$, which must be unique for each tree T*
5. *If $r = 0$, set $\tau \leftarrow \tau || noroot$ and annotate the root: $c_r \leftarrow c_r || theroot$, while all for all other nodes: $c_i \leftarrow c_i || nottheroot$. Otherwise, set $\tau \leftarrow \tau || root$ and annotate all nodes: $c_i \leftarrow c_i || nottheroot$. Sign τ: $\sigma_\tau \leftarrow \mathcal{AGG}.\mathsf{ASign}(\mathrm{sk}_{\mathcal{AGG}}, \tau)$*
6. *Draw a nonce d, $d \xleftarrow{\$} \{0,1\}^\lambda$. For each node n_i: if $i = 0$, set $c_i \leftarrow (c_i || d + \mathrm{depth}(n_i))$, otherwise $c_i \leftarrow (c_i || -1)$. The function $\mathrm{depth} : V \to \mathbb{N}$ returns the distance from the root to the given node $n_i \in T$*
7. *For each node n_i sign $c_i || \tau$: $\sigma_i \leftarrow \mathcal{AGG}.\mathsf{ASign}(\mathrm{sk}_{\mathcal{AGG}}, c_i || \tau)$*
8. *Compress all resulting signatures into the aggregate signature σ_c, along with the signature of τ, i.e., $\sigma_c \leftarrow \mathcal{AGG}.\mathsf{AAgg}(\mathrm{pk}_{\mathcal{AGG}}, \sigma_\tau \cup \{\sigma_i\}_{0<i\leq n})$*
9. *Sign $\mathcal{L}$ using Π, i.e., $(\mathcal{L}, \sigma_\mathcal{L}) \leftarrow \Pi.\mathsf{Sign}(\mathrm{sk}_\Pi, \mathcal{L})$*
10. *Sign $\mathcal{M}$ using Π, i.e., $(\mathcal{M}, \sigma_\mathcal{M}) \leftarrow \Pi.\mathsf{Sign}(\mathrm{sk}_\Pi, \mathcal{M})$*
11. *Output (T, σ_T), where $\sigma_T = (\sigma_c, (\sigma_i)_{0<i\leq n}, \sigma_\tau, \mathcal{L}, \mathcal{M}, \sigma_\mathcal{L}, \sigma_\mathcal{M}, \mathrm{pk}_{\mathcal{AGG}}, \mathrm{pk}_\Pi, \tau)$*

TVerify. *Verifying the signature is similar to generating the signature.*

1. *Check the validity of $\sigma_\mathcal{L}/\mathcal{L}$ and $\sigma_\mathcal{M}/\mathcal{M}$ using $\Pi.\mathsf{Verify}$*
2. *For each node $n_i \in T$, parse c_i as $(m_i, l_i, e_i, a_i, d_i)$*

3. *Traverse T via pre-order: $\forall n_i \in T$, check if $\mathcal{L}_i = l_i$. If not, abort and return* 0
4. *Traverse T via pre-order: $\forall n_j \in T$, check if $\mathcal{M}_j = m_j$. If not, abort and return* 0
5. *Compute $v \leftarrow \mathcal{AGG}.\mathsf{AAggVerf}(\mathrm{pk}, \sigma_T, \{\tau\} \cup \{m_i||l_i||e_i||d_i||a_i||\tau\}_{0<i\leq n})$. If $v = 0$, abort and return* 0
6. *Let r denote the root of the tree T. If $d_r \neq -1$, check for all nodes $n_i \in T \setminus r$, if $\mathrm{depth}(\mathrm{parent}(n_i) = d_i - 1)$, where $\mathrm{parent} : V \to V$ returns the only parent of a given node n_i. If not, abort and return* 0
7. *Parse τ as (τ, h). If $h = \mathsf{noroot}$, check, if the received root's annotion r_a equals "theroot". If so, return* 1*, otherwise return* 0

TShare. *The algorithm $\mathsf{TShare}(T, \mathrm{pk}, \sigma_T, \mathcal{N})$ takes as input the tree T, all public keys* pk *and the signature σ_T, as well as a set of nodes $\mathcal{N} \subseteq T$.*

1. *Remove nodes by setting $T' \leftarrow T \setminus \mathcal{N}$*
2. *If intermediate nodes have been redacted, adjust the edges of the intermediate nodes' successors. In particular, for each node $n_i \in T' \setminus r$ not having a parent, introduce the edge $e_{i,j}$, where n_j is the closest ancestor node not redacted. If the result is not a tree, return $\perp$*
3. *For each $n_i \in \mathcal{N}$, adjust both lists of nonces: $\sigma'_{\mathcal{L}} \leftarrow \Pi.\mathsf{Redact}(\mathrm{pk}_\Pi, \mathcal{L}, i)$, where i is the pre-order number of n_i. And $\sigma'_{\mathcal{M}} \leftarrow \Pi.\mathsf{Redact}(pk_\Pi, \mathcal{M}, j)$, where j is the post-order number of n_i*
4. *Set $\sigma'_c \leftarrow \mathcal{AGG}.\mathsf{AAgg}(\mathrm{pk}_{\mathcal{AGG}}, \sigma_\tau \cup \{\sigma_i\}_{0<i\leq n'})$*
5. *Construct (T', σ'_T), where $\sigma'_T = (\sigma'_c, (\sigma_i)_{0<i\leq n'}, \sigma_\tau, \mathcal{L}', \mathcal{M}', \sigma'_{\mathcal{L}}, \sigma'_{\mathcal{M}}, \mathrm{pk}_{\mathcal{AGG}}, \mathrm{pk}_\Pi, \tau)$*
6. *Verify σ_T: If $\mathsf{TVerify}(T', \mathrm{pk}, \sigma'_T)$ outputs* 0*, abort and return $\perp$*
7. *Output (T', σ'_T)*

3.1 Security

Theorem 1 (The Construction is Secure). *Our construction is secure, if the used RSS is transparent and unforgeable, while the used $\mathcal{AGG}$ is unforgeable.*

Proof. Relegated to App. A.

3.2 Complexity Analysis

For signing our scheme requires $\mathcal{O}(n)$ steps. Each step "involves" each node $n_i \in T$ four times: we assign two random values and we generate two digests. Afterwards, all digests are compressed using $\mathcal{AGG}$, while $\mathcal{L}$ and $\mathcal{M}$ are signed using a order preserving secure RSS Π. We assume that drawing nonces is in $\mathcal{O}(\lambda)$ and therefore constant. The steps performed by Π are not considered in this approximation, as they depend on the actual RSS used, which can be exchanged. Verification "involves" each node in two operations to calculate the digest. Hence, verification is also in $\mathcal{O}(n)$. However, to verify the random values, $\Pi.\mathsf{Verify}$ is called twice. This is also the case during redaction: we simply delete the nodes, while redacting the random values from $\mathcal{L}$ and $\mathcal{M}$ involves two invocations of Π again. Hence, the asymptotic runtime depends on Π, while being at least in $\mathcal{O}(n)$.

4 Modifications to Our Scheme

4.1 Consecutive Redaction Control

To prohibit redaction of several nodes, the ideas introduced in [17] by *Miyazaki* et al. can be applied. Depending on the $\mathcal{AGG}$ used, it is possible to remove a signature from the compressed one. In particular, the *BGLS*-Scheme [4] allows such calculations due to its group-theoretic structure. Hence, to prohibit redaction, the signatures for these nodes are not delivered to the sanitizer. σ_τ is not given to the sanitizer by the signer in the first place. Additional proofs and a more detailed discussion can be found in [17] and [20].

4.2 Restricting to Sanitizers and Accountability

All proposed schemes allow everybody to redact nodes. To limit redaction to explicitly denoted sanitizers the signature σ_T is extended to hold an additional value d. Let $d \leftarrow \text{SIGN}(sk, \tau || \mathcal{CH}(\langle T \rangle))$, where $\langle T \rangle$ is a suitable binary representation of the signed tree T and $\mathcal{CH}$ is a chameleon hash. The values required for $\mathcal{CH}$ need to be delivered with σ_T. Hence, only sanitizers who possess the secret for $\mathcal{CH}$ can alter T without invalidating the signature. This can be enriched further to achieve sanitizer and signer accountability [6]: $\mathcal{CH}$ could be replaced with a tag-based chameleon-hash $\mathcal{CH}_{TAG}$, i.e., the construction of *Brzuska* et al. [6].

5 Implementations and Performance Analysis

We have implemented the scheme by *Kundu* and *Bertino* in the original version [12] and the scheme by *Brzuska* et al. [5]. We did not implement our scheme, since it only differs by a constant factor from the scheme by *Kundu* and *Bertino* in the original version [12], i.e., the underlying RSS Π. The source code used for this evaluation will be made available on request. The tests were performed on a *Lenovo Thinkpad T61* with an *Intel* T8300 Dual Core @2.40 Ghz and 4 GiB of RAM. We ran *Ubuntu* Version 10.04 LTS (64 Bit) and Java version 1.6.0_26-b03. We used 2048-Bit RSA-Keys. The trees signed are n-ary balanced ones with height h. Tab. 1 and 2 show the results for high trees with a low branching factors. Tab. 3 and 4 show the results for flat trees with a high branching factor. This gives a good impression for different use cases and allows determine the specific pros and cons of the schemes.

As shown in the Tables 1-4, all schemes have a comparable runtime for small trees. For binary and ternary trees of increasing depth the scheme by *Brzuska* et al. suffers from the quadratic runtime and becomes unusable, denoted by "-slow-", very fast. Measuring was aborted, if the runtime was greater than 20 minutes. We conclude, that a linear complexity is required to yield useable schemes, as waiting a few minutes to generate σ_T is not acceptable, even if signatures are normally more often verified than generated.

Table 1. *Brzuska* et al.: Low Branching Factor n; Median Runtime in ms

	Generate σ_T			Verify σ_T		
n \ h	10	50	100	10	50	100
2	721	14,319	55,625	24	525	1,910
3	6,856	-slow-	-slow-	241	-slow-	-slow-

Table 2. *Kundu* and *Bertino*: Low Branching Factor n; Median Runtime in ms

	Generate σ_T			Verify σ_T		
n \ h	10	50	100	10	50	100
2	544	2,247	5,293	5	8	10
3	4,319	106,694	369,854	8	165	319

Table 3. *Brzuska* et al.: High Branching Factor n; Median Runtime in ms

	Generate σ_T			Verify σ_T		
n \ h	2	3	4	2	3	4
5	605	2,804	9,607	19	95	333
10	17,195	666,530	-slow-	614	18,838	-slow-

Table 4. *Kundu* and *Bertino*: High branching factor n; Median Runtime in ms

	Generate σ_T			Verify σ_T		
n \ h	2	3	4	2	3	4
5	1,657	5,065	14,132	4	12	22
10	44,756	1,228,738	-slow-	111	864	-slow-

6 Conclusion and Open Questions

After several new attacks presented in this paper, the scheme by *Kundu* and *Bertino* has been found to be insecure with respect to all RSS security properties. Building on the fact that removing an element from a uniformly distributed list of random numbers preserves their distribution and a secure order preserving transparent RSS, the new construction given in this paper can reuse the idea of *Kundu* and *Bertino* that a node's position in a tree is specifiable only by his post- and pre-order traversal numbers. Moreover, our scheme is the first which allows that the signer can decide, if it is allowed to redact parent or intermediate nodes. The new construction is secure against the existing attacks against the scheme by *Kundu* and *Bertino* as given by *Brzuska* et al. [5] and *Samelin* et al. in [20], as well as the two new attacks described in this paper. The paper offers formal proofs of the new construction's security. While all existing secure schemes have a runtime overhead of $\mathcal{O}(n^2)$, our new construction has only an overhead of $\mathcal{O}(n)$, if the underlying order-preserving RSS is in $\mathcal{O}(n)$, i.e., [19]. The work demonstrates that the underlying idea of using traversal numbers to transparently redact nodes combined with a order-preserving RSS can be facilitated to build a simplistic and enhanceable redactable scheme, which is provably secure in terms of transparency, privacy and unforgeability, while being highly efficient and very flexible.

References

1. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order-preserving encryption for numeric data. In: SIGMOD Conference, pp. 563–574 (2004)
2. Ahn, J.H., Boneh, D., Camenisch, J., Hohenberger, S., Shelat, A., Waters, B.: Computing on authenticated data. Cryptology ePrint Archive, Report 2011/096 (2011), `http://eprint.iacr.org/`

3. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable Signatures. In: de Capitani di Vimercati, S., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 159–177. Springer, Heidelberg (2005)
4. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)
5. Brzuska, C., Busch, H., Dagdelen, O., Fischlin, M., Franz, M., Katzenbeisser, S., Manulis, M., Onete, C., Peter, A., Poettering, B., Schröder, D.: Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 87–104. Springer, Heidelberg (2010)
6. Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of Sanitizable Signatures Revisited. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 317–336. Springer, Heidelberg (2009)
7. Canard, S., Jambert, A.: On Extended Sanitizable Signature Schemes. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 179–194. Springer, Heidelberg (2010)
8. Chang, E.-C., Lim, C.L., Xu, J.: Short Redactable Signatures Using Random Trees. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 133–147. Springer, Heidelberg (2009)
9. Izu, T., Kunihiro, N., Ohta, K., Sano, M., Takenaka, M.: Sanitizable and Deletable Signature. In: Chung, K.-I., Sohn, K., Yung, M. (eds.) WISA 2008. LNCS, vol. 5379, pp. 130–144. Springer, Heidelberg (2009)
10. Johnson, R., Molnar, D., Song, D., Wagner, D.: Homomorphic Signature Schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 244–262. Springer, Heidelberg (2002)
11. Klonowski, M., Lauks, A.: Extended Sanitizable Signatures. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296, pp. 343–355. Springer, Heidelberg (2006)
12. Kundu, A., Bertino, E.: Structural Signatures for Tree Data Structures. In: Proc. of PVLDB 2008, New Zealand. ACM (2008)
13. Kundu, A., Bertino, E.: CERIAS Tech Report 2009-1 Leakage-Free Integrity Assurance for Tree Data Structures (2009)
14. Kundu, A., Atallah, M.J., Bertino, E.: Leakage-free redactable signatures. In: CODASPY, pp. 307–316 (2012)
15. Kundu, A., Bertino, E.: How to authenticate graphs without leaking. In: EDBT, pp. 609–620 (2010)
16. Kundu, A., Bertino, E.: Structural signatures: How to authenticate trees without leaking. Technical report, Purdue University (June 2010)
17. Miyazaki, K., Hanaoka, G., Imai, H.: Digitally signed document sanitizing scheme based on bilinear maps. In: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2006, pp. 343–354. ACM, New York (2006)
18. Pöhls, H.C., Samelin, K., Posegga, J.: Sanitizable Signatures in XML Signature — Performance, Mixing Properties, and Revisiting the Property of Transparency. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 166–182. Springer, Heidelberg (2011)
19. Pöhls, H.C., Samelin, K., Posegga, J., de Meer, H.: Length-hiding redactable signatures from one-way accumulators in $\mathcal{O}(n)$ (mip-1201). Technical report, University of Passau (April 2012)

20. Samelin, K., Pöhls, H.C., Bilzhause, A., Posegga, J., de Meer, H.: Redactable Signatures for Independent Removal of Structure and Content. In: Ryan, M.D., Smyth, B., Wang, G. (eds.) ISPEC 2012. LNCS, vol. 7232, pp. 17–33. Springer, Heidelberg (2012)
21. Steinfeld, R., Bull, L., Zheng, Y.: Content Extraction Signatures. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 285–304. Springer, Heidelberg (2002)

A Proofs

Theorem 2 (The Construction is Transparent). *Our construction is transparent and therefore also private [5].*

Proof. $\mathcal{L}$, $\mathcal{M}$, the public keys $pk_{\mathcal{AGG}}$ and pk_{Π} and the tag τ do not leak any information about the tree T. The lists $\mathcal{L}$ and $\mathcal{M}$ contain uniformly distributed random numbers. Even on redaction of a tree node, we only remove elements from a uniformly distributed list of randoms, which still results in a uniformly distributed list of randoms. If the tree is annotated with d, nothing can be derived either, since d is chosen from a uniform distribution and the tree grows at most linearly in $|V|$. Hence, we only need to show that $\sigma_{\mathcal{L}/\mathcal{M}}$ and σ_c together imply transparency. A successful attack on transparency would show that either Π and/or the aggregate signature scheme $\mathcal{AGG}$ is not transparent. To avoid duplicate work, we relegate the reader to [17] and [20], where the proofs for transparency of σ_c can be found. For $\sigma_{\mathcal{L}/\mathcal{M}}$, we show how an adversary would also break the transparency of the underlying RSS Π: Assume an efficient adversary $\mathcal{A}_{tra}$ which wins the transparency game of our scheme. Using $\mathcal{A}_{tra}$ we construct another adversary $\mathcal{B}_{tra}$ to break the transparency of Π in the following way: For any calls to $\mathcal{O}^{\mathsf{TSign}}$ resp. $\mathcal{O}^{\mathsf{Adapt/Sign}}$, $\mathcal{B}_{tra}$ genuinely returns the answers of its own oracle. Eventually, $\mathcal{A}_{tra}$ outputs its guess b^*. b^* is then outputted by $\mathcal{A}_{tra}$ as its own guess. We now need to argue that, due to the fact that $\mathcal{AGG}$ is information-theoretically transparent, $\mathcal{B}_{tra}$'s probability of success equals the one of $\mathcal{A}_{tra}$. Only Π could have leaked this information, as the lists contain just uniformly distributed nonces and redactions of elements in that list again lead to a uniformly distributed list. Hence, $\mathcal{B}_{tra}$ wins with the same probability as $\mathcal{A}_{tra}$.

Theorem 3 (The Construction is Unforgeable). *Our construction is unforgeable.*

Proof. We assume that no tag-collisions occur, winning the unforgeability game in a trivial manner. Note, that we do not need an induction over the tree, as we transform it into two lists, which are uniquely determined. Let the algorithm winning the unforgeability game be denoted as $\mathcal{A}$. Our scheme's security relies upon the security of $\mathcal{AGG}$ and Π. Given the game in Fig. 1, we can derive that a forgery must fall in at least one of the following cases:

Case 1: A value protected by σ_c has never been signed by the oracle
Case 2: The value protected by σ_c has been signed, but $T^* \not\subseteq \text{span}_{\vdash}(T)$. In other words: The tree T^* protected by σ_T is not in the transitive closure of any tree for which a signature was queried. This case has to be divided as well:
Case 2a: $T \notin \text{span}_{\vdash}(T^*)$
Case 2b: $T \in \text{span}_{\vdash}(T^*)$

We can construct an adversary $\mathcal{B}$, which breaks the unforgeability of the BGLS-Scheme, if an adversary $\mathcal{A}$ with a non-negligible advantage ϵ exists, winning our unforgeability game. To do so, $\mathcal{B}$ uses $\mathcal{A}$ as a black box. For every signature query $\mathcal{A}$ requests, $\mathcal{B}$ forwards the queries to its signing oracle $\mathcal{O}^{TSign}$ and genuinely returns the answers to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs (T^*, σ_T^*), where $T^* = (\sigma_c^*, \{\sigma_i\}_{0<i\leq n^*}, \sigma_\tau^*, \mathcal{L}^*, \mathcal{M}^*, \sigma_{\mathcal{L}}^*, \sigma_{\mathcal{M}}^*, pk_{\mathcal{AGG}}^*, pk_{\Pi}, \tau^*)$. Given the transcript of the simulation, $\mathcal{B}$ checks, if the outputted tuple is a trivial "forgery", i.e., just an allowed redaction. If so, $\mathcal{B}$ aborts the simulation. If $\mathcal{B}$ does not abort, we have to consider the following constellations: If σ_c^* contains messages never signed, $\mathcal{B}$ outputs (σ_c^*) along with forged strings, which can easily be extracted. If $T \notin \text{span}_{\vdash}(T^*)$, we have to consider two cases: (1) the values protected by $\sigma_{\mathcal{L}}$ resp. $\sigma_{\mathcal{M}}$ have been altered or (2) the strings protected by σ_T were modified. In either case, this allows to break the unforgeability of either Π or $\mathcal{AGG}$. The corresponding forgeries can easily be extracted. If $T \in \text{span}_{\vdash}(T^*)$, both Π and $\mathcal{AGG}$ must have been forged, since new elements are now contained in the aggregate or the RSS. As before, the forgeries can be extracted given the transcript.