Redactable Signatures for Independent Removal of Structure and Content (Extended)

Kai Samelin$^{\$,\star}$, Henrich C. Pöhls$^{\$,\star\star}$, Arne Bilzhause$^{\$}$, Joachim Posegga$^{\$}$, Hermann de Meer$^{\#}$

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 a provably secure redactable signature scheme allowing to independently redact structure and content. We identify the problems when structure is not separated from content, resulting in an attack on the scheme proposed at VLDB '08 by *Kundu* and *Bertino*. The attack allows for changing the semantic meaning of a given tree. We introduce a rigid security model, including consecutive redaction control, to formalize the required behaviour of our scheme. Moreover, we present first performance evaluations of our implementation to demonstrate the practical use of the presented scheme.

Keywords: Structural Integrity, Redactable Signatures, Implementation

1 Introduction

A redactable signature scheme (RSS) allows a third party to remove parts of a signed document m without invalidating its protecting signature σ. This action can be performed without involvement of the original signer. In more detail, a RSS allows a third party to replace parts of the original document with $\perp$, a special symbol indicating that a redaction took place. As a result, the verifier only sees a blinded version of the document, while still being able to verify that the remaining subdocuments are still valid and authentic. The RSSs we will consider allow for public redactions, i.e. no private keys are required to perform a redaction. Moreover, a RSS can allow to prohibit a consecutive third party to remove certain parts, a property named Consecutive Redaction Control [20]. The notation we use for a document is $m = m[1]||\ldots||m[\ell]$. We will call each $m[i]$ a submessage, where $\ell \in \mathbb{N}^+$ is the number of submessages and $||$ a concatenation. Current schemes just allow to redact subdocuments. In particular, no existing

*Operationelles Programm des Europäischen Fonds für regionale Entwicklung (EFRE) im Ziel "Regionale Wettbewerbsfähigkeit und Beschäftigung" Bayern 2007-2013

**Is funded by BMBF (FKZ:13N10966) and ANR as part of the ReSCUeIT project

scheme allows redacting the ordering or other structures, which carry information as well. This may not be sufficient in some cases, which is shown show next.

Examples and Benefits of Redacting Structure Separately. Structured data comes in many forms, for example XML-Schemata describe the structure (and often implicitly the semantics) of tree-structured XML-Documents. Even in linear documents, e.g. a text file, the order of subdocuments is important, e.g. the ordering of chapters in a book. *Liu* et al. introduced the paradigm of separating structural integrity and content integrity in [18] without considering RSS. This implicit information stored inside the structure of a document leads to our attack on the *Kundu*-Scheme.

The "digital document sanitization problem", as introduced by *Miyazaki* et al. in [19], assumes that the signing process itself cannot be altered. This may happen, if the signer is not reachable anymore, or must not know which parts of the document are passed to third parties. Consider the following two examples clarifying why one needs to be able to redact structure: (1) In an university the exam results are published in a list. The list first gives the student's name ($m[1]$) and the grade ($m[2]$), then the next student's name ($m[3]$) and then the grade ($m[4]$) and so on. Imagine the list being ordered by grades, i.e. the student with the best grade is at the beginning. Thus, the subdocuments $m[i]$ hold information about either a grade or a student's name, while the ordering carries the information "better-than". We only want a signed list of all the students' name who took part in the exam. Hence, we need to redact all information about grades from this list. A redactable scheme allows deleting the grades, i.e. all $m[i] \leftarrow \perp$, where i is even. Using a transparent [1, 4, 5] RSS would also remove the trace that some parts where removed. In particular, $\perp$ would not be visible anymore. However, the original ordered relation of the remaining subdocuments is still present, invading privacy. Hence, we also need to redact the ordering among them. Current schemes are not able to redact this information; they require that the ordering cannot be redacted or work only on structureless sets [20].
(2) From a business point of view, we can derive our second example: The sales department provides a monthly overview over all sent invoices to an external auditor. However, the order in which a company sends out its invoices may leak some business critical information. Further imagine that the list must be ordered in a monthly manner, i.e. the invoices need to be grouped by month, while the internal ordering must not be made public to protect trade-secrets. This requires the RSS to explicitly sign each subdocument relation separately to allow removal of only this information at a later stage.

There are many other application scenarios, e.g. one can redact hierarchies within companies, or explicitly sign partially ordered sets, which is useful for information flow models [24]. Many more examples for the use of RSS are given in the original works, our main motivation is to maintain privacy or protect trade secrets. Hence, the RSS is required to be transparent, meaning the fact that redaction has occurred must not be known by a verifying third party. Loss of transparency would decrease the value of the redacted information, e.g. a summarized report redacted by the PR department is less valuable than a summarized report where redactions by the PR department are hidden. Hence, we require a usable scheme to be transparent and allow redactions of structure to cater for such constellations.

State of the Art and Current Limitations. The concept of removing parts from signed data without invalidating the signature was initially introduced as "content extraction signature" by *Steinfeld* et al. in [25] resp. in [12] by *Johnson* et al. Their inspiring work has lead to many RSS constructions in the last years [8, 20, 21]. The schemes have been extended to work on tree-structured data [4, 15, 16] and on arbitrary graphs [17]. However, the schemes proposed in [15] and [16] suffer from an attack vector changing the semantics of the tree. We will present this attack in this paper. A related concept are sanitizable signature schemes (SSS), introduced by *Ateniese* et al. in [1], where the choice of values for a specific submessage $m[i]$ is not per se limited to $m[i]$ or $\perp$, but to arbitrary strings $\subset \{0,1\}^*$. Limiting third parties to certain values is a well-known field [7, 13, 23].

Recently, *Brzuska* et al. defined and formalized a set of desired properties for redactable tree-structured documents in [4]. In this paper, we extend these definitions towards linear documents with separate structure redaction. Most of the schemes proposed up to now are not transparent, i.e. one can see that a third party redacted something [10, 12, 21, 25, 26]. Furthermore, this also impacts on privacy, as already noted in [16] and [20]. The scheme introduced in [20] also suffers from several problems: (1) It is just useable for sets, which means that the ordering, what we call structure, is not protected. (2) It does not allow multi-sets, i.e. each element, resp. subdocument must be unique. (3) This fact is not checked during their verification algorithm, hence their scheme is forgeable, i.e. by copying existing elements. The only other provably transparent schemes, i.e. [4] and [8], require $\mathcal{O}(n^2)$ operations as well, but their only gain is transparency, hence coming very costly. We have the same complexity, but allow much more freedom, i.e. removing structure and allowing consecutive redaction control.

Our Contribution and Organization. In this paper, we present the first transparent RSS for ordered linear documents which redacts content and structure separately. We introduce a precise formal model for this new paradigm. This model will also include consecutive redaction control, which has been used in several papers [11, 20, 21], but has not yet been formalized rigorously. We will denote the set of admissible redactable entities as ADM, following the notation of [5]. Moreover, we present an attack on the *Kundu*-Scheme in Sect. 2, allowing to exploit the implicit semantic meaning of structure. Furthermore, we propose a concrete provably secure RSS construction, which will meet all of our security

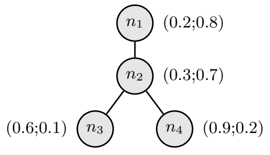

Fig. 1. Original Tree T

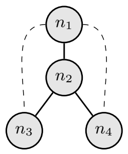

Fig. 2. Transitive Closure of T, i.e. $\text{span}_{\models}(T)$

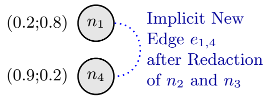
(0.2;0.8)
n_1
(0.9;0.2)
n_4
Implicit New Edge $e_{1,4}$ after Redaction of n_2 and n_3

Fig. 3. Redacted Tree T'

requirements stated in Sect. 4. Our scheme is also secure against the probabilistic attacks on transparency described by *Brzuska* et al. in [4]. Existing schemes fell victim to this attack: The scheme by *Kundu* and *Bertino* [16] lacks provably transparency since it uses ordered random numbers [4]. Also the schemes by *Izu* et al. in [11] are suspect to this attack. This is a result of the proofs given for the *Kundu*-Scheme in [4]. Hence, we introduce a way to sign the ordering of all submessages $m[i]$ in this paper. Our solution will sign each "left-of" relation to allow transparent redactions, as already proposed by *Chang* et al. in [8] and by *Brzuska* et al. in [4]. In particular, each pair $(m[i], m[j])$ gets signed, where $0 < i < j \leq \ell$. However, their schemes do not allow to redact any structure. Our solution requires $\mathcal{O}(n^2)$ operations for signing and verification. This is similar to [4] and [8]. A more detailed theoretical cost analysis is provided as well, while a performance comparison of our implementation using real data can be found in Sect. 5. Additional preliminaries and our extended model are presented in Sect. 3. Formal proofs of the security and correctness are in the appendix.

2 Attacking the *Kundu*-Scheme

The only RSS for trees able to redact non-leafs is the *Kundu*-Scheme, introduced in [15] and revised in [16]. Their scheme builds upon the idea that a third party having the pre- and post-order traversal numbers of all nodes contained in a tree T is always able to correctly reconstruct T. Hence, signing each node $n_i \in T$ along with both numbers and the content is enough to protect T. To make the scheme transparent, these traversal numbers are randomized in an order-preserving manner, which does not have an impact on the reconstruction algorithm, which just checks for greater than relations [15, 16]. Thus, verification is straight forward — with one additional step: A verifier has to check if all nodes are in the correct order using the traversal numbers. 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. However, as shown in the introduction, the structure itself does carry information as well: Assume that one removes n_2 and n_3 from T, as depicted in Fig. 3. This allows to add a new edge $e_{1,4}$, which has not explicitly been present in the original tree T. This implies, that the tree $T^{\mathcal{A}} = (\{n_1, n_4\}, \{e_{1,4}\})$ is valid in terms of the signature. For more detail we 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 $(4, 3, 2, 1)$. The randomization step may transform them into $(0.2, 0.3, 0.6, 0.9)$ and $(0.8, 0.7, 0.2, 0.1)$ resp. Hence, the node n_1 has a structural position of $\rho_1 = (0.2; 0.8)$. For n_2, n_3 and n_4 this is done accordingly. We redact n_2, an intermediate node of n_4, and n_3. For the redacted tree in Fig. 3, the traversal-numbers are still in the correct order. Hence, the signature verifies. *Kundu* and *Bertino* neither prohibit nor exclude the redaction of intermediate nodes, but claim this is a useful property [16]. This behaviour is problematic, as we will show next. The *Kundu*-Scheme signs the transitive closure of the tree T, as depicted in Fig. 2. This is a very weak form of structural integrity protection and allows some semantic attacks: Consider a hierarchical structure of treatments inside a medical database, i.e. treatments consists of treatments, e.g. a chemotherapy consists of giving drugs against cancer and additional prophylactic drugs to avoid infections, codified into the tree's structure. If the cancer drugs and chemotherapy node is redacted, the only treatment node left is the prophylactic drug one. This does neither destroy any XML-Schemata nor can it be detected by humans. This behaviour is not acceptable, though it may have its application, e.g. to redact hierarchies within a company. However, we argue that the places where this is allowed must be explicitly denoted by the signer to avoid the mentioned attack. It remains on open question how to construct a secure scheme, which allows such constellations, along with a suitable security model, since the one introduced in [4] just allows to redact leafs.

3 Preliminaries, Notations and Security Properties

Basically, we have the same requirements as *Brzuska* et al. [4], i.e. unforgeability, privacy and transparency. However, we need to adjust the definitions as we treat structure as a redactable entity and to make statements about linear documents instead of trees. We require that the splitting of m into the subdocuments $m[i]$, along with the order, is efficiently reconstructable from any received m.

Definition 1 (RSS Algorithms for Content and Structure Redaction). *A RSS which allows separate redaction of content and structure, with consecutive redaction control, consists of five efficient algorithms. In particular,* $\mathcal{RSS} := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact}, \mathsf{Close})$ *such that:*

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

Sign. *The algorithm* Sign *outputs the signature* σ *on input of the secret key* sk *and the document* m. *It outputs* $(m, \sigma) \leftarrow \mathsf{Sign}(\mathrm{sk}, m)$.

Verify. *The algorithm* Verify *outputs a bit* $d \in \{0, 1\}$ *indicating the correctness of the signature* σ, *w.r.t.* pk, *protecting* m. *In particular:* $d \leftarrow \mathsf{Verify}(\mathrm{pk}, m, \sigma)$

Redact. *The algorithm* Redact *takes as input the document* m*, the public key* pk *of the signer, the signature* σ *and description of the redaction* $\textsc{mod}$ *containing either a submessage* $m[i]$ *or a binary relation* $m[i,j]$ *that shall be redacted. Calling* Redact *sequentially allows to redact more relations and submessages. The algorithm outputs* $(m',\sigma') \leftarrow \mathsf{Redact}(\mathrm{pk}, m, \sigma, \textsc{mod})$*, where* $m' \leftarrow \textsc{mod}(m)$ *denotes the alteration of* m *w.r.t.* $\textsc{mod}$*.* $\textsc{mod}$ *may contain more than one modification in our notation. We require that* ADM*, which denotes the entities of* m *admissible to be redacted, is always correctly recoverable from* (m,σ)*. The algorithm doing so will be described as:*

Sanitizable. *On input of a valid message/signature pair* (m,σ)*,* $\mathsf{Sanitizable}$ *outputs* ADM*:* $\mathrm{ADM} \leftarrow \mathsf{Sanitizable}(m,\sigma)$*.*

We will treat $\mathsf{Sanitizable}$ *as part of* Redact *and not as a stand alone algorithm. Note,* Redact *allows public redactions, since just the public key* pk *of the signer is required.*

Close. *The algorithm* Close *alters* σ *on input of* m*, the public key* pk *of the signer, the signature* σ *and a sanitization control description* $\textsc{mod}_c$ *which contains the entities subject to redaction control. The algorithm outputs* $(m,\sigma') \leftarrow \mathsf{Close}(\mathrm{pk}, m, \sigma, \textsc{mod}_c)$*.* $\textsc{mod}_c$ *may contain many modifications.* Close *does not change the message* m *itself, but* $\mathsf{Sanitizable}(m,\sigma') = \mathsf{Sanitizable}(m,\sigma) \setminus \textsc{mod}_c$*. This algorithm can be called by the signer and any other third party. This enables the signer to close parts of* m *prior to distributing.*

Signing each “left-of” relation is enough to protect the structure of an ordered document. This has already been utilized and proven in [8] and [4]. The signer is able to redact parts of the document as well, since a RSS allows public redaction. Hence, all parties can sanitize and close the document, which includes the signer and the final recipient as well.

3.1 Security Properties

We now define the required security properties of RSS. These have already been identified in [4] for trees. Therefore, we will adapt and extend their notion for our needs. We will denote the transitive closure of a message m, w.r.t. to Redact, as $\mathrm{span}_{\models}(m)$, which is derived from [8]. We denote a redaction of a submessage $m[i]$ as $m \setminus m[i]$. A redaction of a relation between $m[i]$ and $m[j]$ will be denoted as $m \setminus m[i,j]$. Note, the following definitions address only the information a third party can derive from the signature; e.g., if obvious redactions took place, it may be trivial to decide whether something has been redacted. We will use the notation $\sqsubseteq$ to express a subset relation in terms of submessages and submessage relations.

```
Experiment $\mathsf{Unforgeability}^{\mathsf{RSS}}_{\mathcal{A}}(\lambda)$
    $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
    $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pk)$
        let $i = 1, 2, \ldots, q$ index the queries
    return 1 iff
        $\mathsf{Verify}(pk, m^*, \sigma^*) = 1$ and
        $\forall i, 1 \leq i \leq q : m^* \notin \mathrm{span}_{\models}(m_i)$
```

Fig. 4. Unforgeability

```
Experiment $\mathsf{Privacy}^{\mathsf{RSS}}_{\mathcal{A}}(\lambda)$
    $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
    $b \xleftarrow{\$} \{0, 1\}$
    $d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot),\mathsf{LoRRedact}(\ldots,sk,b)}(pk)$
        where oracle LoRRedact
        for input $m_0, m_1, \mathrm{MOD}_0, \mathrm{MOD}_1$:
        if $\mathrm{MOD}_0(m_0) \neq \mathrm{MOD}_1(m_1)$, return $\perp$
        $(m, \sigma) \leftarrow \mathsf{Sign}(sk, m_b)$
        return $(m', \sigma') \leftarrow \mathsf{Redact}(pk, m, \sigma, \mathrm{MOD}_b)$.
    return 1 iff $b = d$
```

Fig. 5. Privacy

Unforgeability. No one should be able to compute a valid signature on a document outside the transitive closure $\mathrm{span}_{\models}(m)$, without having access to the secret key *sk*. That is, even if an outsider can request signatures on different documents, it remains impossible to forge a signature for a new document. This is analogous to the standard unforgeability requirement for other signature schemes. We say that a *RSS* is unforgeable, iff for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 4 returns 1, is negligible (as a function of λ).

Privacy. No one should be able to gain any knowledge about redacted parts without having access to them. This is similar to the standard indistinguishability notion for encryption schemes. We say that a *RSS* for documents is private, iff for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 5 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ).

Transparency. The verifier should not be able to decide whether a signature has been created by the signer, or through the redaction algorithm Redact. This means, that a party cannot decide whether a freshly signed or a blinded version where some parts have already been redacted has been received. We say that a *RSS* is transparent, iff for any efficient (PPT) adversary $\mathcal{A}$, the probability that the game depicted in Fig. 6 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ).

Disclosure Secure. No one should be able to redact parts of a document which are not part of ADM. This is analogous to the immutability requirement for SSS [5]. Note, in [20] *Miyazaki* et al. merged this with unforgeability. However, for unforgeability any message is enough to break the game; for disclosure security, an adversary has two possibilities: Either it is able to redact a part which is subject to redaction control or is able to alter ADM, such that the disclosure control is reversed. Therefore, the games are slightly different. Additionally, our game is stricter as the adversary can choose the parts of the message to be

```
Experiment $\mathsf{Transparency}_{\mathcal{A}}^{\mathsf{RSS}}(\lambda)$
    $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
    $b \xleftarrow{\$} \{0,1\}$
    $d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot),\mathsf{Sign/Redact}(\ldots,sk,b)}(pk)$
        where oracle $\mathsf{Sign/Redact}$ for input $m$, $\textsc{mod}$:
            if $\textsc{mod}(m) \notin \mathrm{span}_{\models}(m)$, return $\perp$
            if $b = 0$:   $(\sigma, m) \leftarrow \mathsf{Sign}(sk, m)$,
                          $(\sigma', m') \leftarrow \mathsf{Redact}(pk, \sigma, m, \textsc{mod})$
            if $b = 1$:   $m' \leftarrow \textsc{mod}(m)$
                          $(m', \sigma') \leftarrow \mathsf{Sign}(sk, m')$,
        finally return $(m', \sigma')$.
    return 1 iff $b = d$
```

Fig. 6. Transparency

```
Experiment $\mathsf{DisclosureSecure}_{\mathcal{A}}^{\mathsf{RSS}}(\lambda)$
    $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
    $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{RSign}(sk,\cdot)}(pk)$
        where oracle $\mathsf{RSign}$ for input $m$, $\textsc{mod}_c$:
            $(\sigma, m) \leftarrow \mathsf{Sign}(sk, m)$
            return $\mathsf{Close}(pk, m, \sigma, \textsc{mod}_c)$
    return 1 iff
        let $i = 1, 2, \ldots, q$ index the queries
        let $\mathrm{ADM}_i \leftarrow \mathsf{Sanitizable}(m_i, \sigma_i)$ and
        let $\mathrm{ADM}^* \leftarrow \mathsf{Sanitizable}(m^*, \sigma^*)$ and
        $\mathsf{Verify}(pk, m^*, \sigma^*) = 1$ and
            $\exists i : m^* \in \mathrm{span}_{\models}(m_i) \wedge m_i \setminus m^* \nsubseteq \bigcup_{0<i\leq q} \mathrm{ADM}_i$ or
            $\exists i : \mathrm{ADM}^* \supset \mathrm{ADM}_i \wedge \forall i : \mathrm{ADM}^* \nsubseteq \bigcup_{0<i\leq q} \mathrm{ADM}_i$
```

Fig. 7. Disclosure Secure

subject to disclosure control. We say that a RSS is disclosure secure, iff for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 7 returns 1, is negligible (as a function of λ).
Next, we will describe the needed primitives for our scheme.

3.2 Aggregate Signatures and Bilinear Pairings

Aggregate signatures ($\mathcal{AGG}$) have been introduced by *Boneh* et al. in [3]. The basic idea is as follows: Given ℓ signatures σ_i, $0 < i \leq \ell$, one constructs a compressed signature σ which contains all signatures σ_i. This allows verifying all given signatures σ_i by verifying σ. The scheme can be constructed as follows: 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 $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ 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. [3] (BGLS-Scheme) with public aggregation consists of five efficient algorithms. Especially:*

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

AKeyGen. *The algorithm KeyGen outputs the public and private key of the signer,* $\mathrm{sk} \xleftarrow{\$} \mathbb{Z}/q\mathbb{Z}$ *denote the signer's private key and* $\mathcal{H}_k : \{0,1\}^* \rightarrow \mathbb{G}_1$ *an ordinary cryptographic hash-function from the family* $\mathcal{H}_K$ *and set* $Q \leftarrow g^{\mathrm{sk}}$*, where* g *is a generator of* $\mathbb{G}_1$*. Set the public parameters and key* $\mathrm{pk} \leftarrow (g, Q, \mathbb{G}_1, \mathbb{G}_T, \mathcal{H}_k, \hat{e})$*. Output* $(\mathrm{pk}, \mathrm{sk})$.

```
Experiment $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathcal{AGG}}(\lambda)$
    $(pk_c, sk_c) \leftarrow \mathsf{AKeyGen}(1^\lambda)$
    $(M^*, \sigma^*, pk^*) \leftarrow \mathcal{A}^{\mathsf{ASign}(sk,\cdot)}(pk)$
    Let $\omega$ be the index of $pk_c$ in $pk^* = (pk_1^*, \ldots, pk_\ell^*)$
    and $M^* = (m_1^*, \ldots, m_\ell^*)$.
    return 1 iff
        $\mathsf{AAggVerf}(pk^*, \sigma^*, M^*) = 1$ and
        $pk_\omega \in pk^*$ and $m_\omega^*$ has never
        been queried under $pk_\omega$
```

Fig. 8. Aggregate Unforgeability

```
Experiment $\mathsf{ExtractSecure}_{\mathcal{A}}^{\mathcal{AGG}}(\lambda)$
    Let $k > 1$ denote some security parameter
    For: $i = 1..k$: $(pk_{c_i}, sk_{c_i}) \leftarrow \mathsf{AKeyGen}(1^\lambda)$
    $(pk^*, \sigma^*, M^*) \leftarrow \mathcal{A}^{\mathsf{AAggSign}(sk_c, \cdots)}(pk_{c_i})$
        where oracle AAggSign for input $(m_1, \ldots, m_k)$
            For: $i = 1..k$: $(\sigma_i, m_i) \leftarrow \mathsf{ASign}(sk_{c_i}, m_i)$
            return $\mathsf{AAgg}((pk_{c_1}, \ldots, pk_{c_k}), (\sigma_1, \ldots, \sigma_k))$
    return 1
        let $j = 1, 2, \ldots, q$ index the queries
        $\mathsf{Verf}(pk^*, \sigma^*, M^*) = 1$ and
        $pk^* \subseteq pk_c$ and $(pk^*, \sigma^*, M^*)$ is non-trivial, i.e.:
        $\sigma^*$ is not in the transitive closure of the
        messages queried to the oracle
```

Fig. 9. k-Element-Aggregate-Extraction

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

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

AAgg. *To aggregate ℓ signatures σ_i, protecting m_i, into an aggregated signature σ, the aggregator computes $\sigma \leftarrow \prod_{i=1}^{\ell} \sigma_i$, denoted as AAgg$(\mathrm{pk}, \mathcal{S})$, where $\mathcal{S}$ is a set of signatures signed using the same public parameters. Note: This can be done by untrusted parties and without knowing the private keys.*

AAggVerf. *To verify an aggregated signature σ, a verifier checks whether $\hat{e}(\sigma, g) \overset{?}{=} \prod_{i=1}^{\ell} \hat{e}(\mathcal{H}_k(m_i), Q_i)$ holds, on input of σ, a list of public keys $\mathrm{pk} = (\mathrm{pk}_1, \ldots, \mathrm{pk}_\ell)$ and a list of signed messages $M = (m_1, \ldots, m_\ell)$. To improve efficiency, the right side can be rewritten as $\hat{e}(\prod_{i=1}^{\ell} \mathcal{H}_k(m_i), Q)$, if we just use one public key, Q, which allows this improvement. Using just one public key also has the advantage that we are sure that just one signing key is used. We denote the algorithm as $d \leftarrow$ AAggVerf$(\mathrm{pk}, \sigma, \{m_i\}_{0<i\leq\ell})$.*

As usual, the correctness requirements should hold. Formal proofs of those can be found in [3]. We require the expected security properties to hold, i.e. unforgeability under chosen message attacks. The proofs can also be found in [3]. We explicitly assume that splitting up an aggregate signature is not feasible, as shown for the BGLS-Scheme in [9]. However, we require that the attacker has access to a signing oracle where it can chose the messages. For the BGLS-Scheme, this has already been assumed in [20], but is not stated formally in [3]. In particular, we require that the probability that the games depicted in Fig. 8 and Fig. 9 return 1 is negligible. Moreover, it is required that, if a third party knows a contained signature, it can build an inverse and actually remove the signature from the aggregate. We will denote the removal of σ_i from σ as $\sigma' \leftarrow \sigma \setminus \sigma_i$. For the BGLS-Scheme [3] this means: $\sigma' \leftarrow \sigma \cdot \sigma_i^{-1}$. We will use this behaviour to obtain secure consecutive redaction control.

4 *RSS* Construction using Aggregate Signatures

Our construction is based upon the defined $\mathcal{AGG}$. The construction introduced will be generic, we give an instantiation afterwards. It extends the scheme introduced by *Miyazaki* et al. in [20] without inheriting its flaws and limitations. Using aggregating signatures over accumulating hashes [2] resp. distributing many signatures has three advantages: (1) We can introduce consecutive redaction control; (2) we are information theoretically secure, both in terms of transparency and privacy; (3) we speed up the verification procedure as well.

4.1 High-Level Description of Our Construction

Construction 1 (RSS) *Our construction makes use of an aggregating signature scheme $\mathcal{AGG}$ as defined earlier. We will explain every algorithm in detail next. Note, this is a high-level description; an instantiation based on the BGLS-Scheme is given in Sect. 4.2.*

Key Generation. *The key pair generation algorithm KeyGen outputs the key pair $(\mathrm{sk}, \mathrm{pk})$, i.e.: $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{AKeyGen}(1^\lambda)$, i.e. it uses the key pair of the underlying $\mathcal{AGG}$.*

Signing. *To sign m, where all "left-of" relations $m[i,j]$ can be derived from, perform the following steps:*

1. *Choose a nonce τ, i.e. τ must be unique for each document signed. The tag is needed to avoid adding subdocuments from other documents signed with the same secret key* sk
2. *Sign τ, i.e. $\sigma_\tau \leftarrow \mathsf{ASign}(\mathrm{sk}, \tau)$*
3. *Draw ℓ pair-wise distinct nonces r_i from a uniform distribution. These are needed to prevent an adversary from aggregating a contained entity twice*
4. *Append each r_i to the corresponding subdocument $m[i] \sqsubseteq m$, then append τ and sign the resulting string, i.e. $\sigma_i \leftarrow \mathsf{ASign}(\mathrm{sk}, \tau || r_i || m[i])$*
5. *Sign each existing tagged "left-of" relation: $\sigma_{i,j} \leftarrow \mathsf{ASign}(\mathrm{sk}, \tau || r_i || r_j)$, for all $0 < i < j \leq \ell$, if $m[i,j] \sqsubseteq m$*
6. *Aggregate each generated signature, i.e:*
$$\sigma_c \leftarrow \mathsf{AAgg}(\mathrm{pk}, \sigma_\tau \cup \{\sigma_i \mid m[i] \sqsubseteq m\} \cup \{\sigma_{i,j} \mid m[i,j] \sqsubseteq m\})$$
7. *Output $\sigma = (\sigma_c, \tau, \{\sigma_i \mid m[i] \sqsubseteq m\}, \{\sigma_{i,j} \mid m[i,j] \sqsubseteq m\}, \{r_i \mid m[i] \sqsubseteq m \vee m[i,j] \sqsubseteq m \vee m[j,i] \sqsubseteq m\})$*

Note: This algorithm already allows to sign partially ordered sets by not requiring all relations; this is necessary to maintain privacy and transparency.

Redact. *To redact a subdocument $m[i]$, the following steps are performed:*

1. *Check σ's validity using* Verify. *If the signature is not valid, return $\perp$*
2. *If $m[i] \not\sqsubseteq m$, return $\perp$*
3. *Set $m' = m \setminus m[i]$. Note, this does not redact the submessage's relations*
4. *Calculate: $\sigma'_c = \sigma_c \setminus \sigma_i$*
5. *Output $\sigma' = (\sigma'_c, \tau, \{\sigma_i \mid m'[i] \sqsubseteq m'\}, \{\sigma_{i,j} \mid m'[i,j] \sqsubseteq m'\}, \{r_i \mid m'[i] \sqsubseteq m' \vee m'[i,j] \sqsubseteq m' \vee m'[j,i] \sqsubseteq m'\})$*

To redact a relation $m[i,j]$, the third party has to perform the following steps:

1. *Check σ's validity using* Verify. *If the signature is not valid, return $\perp$*
2. *If $m[i,j] \not\sqsubseteq m$, return $\perp$*
3. *Set $m' = m \setminus m[i,j]$. Note, this does not redact submessages m_i nor m_j*
4. *Calculate: $\sigma'_c = \sigma_c \setminus \sigma_{i,j}$*
5. *Output $\sigma' = (\sigma'_c, \tau, \{\sigma_i \mid m'[i] \sqsubseteq m'\}, \{\sigma_{i,j} \mid m'[i,j] \sqsubseteq m'\}, \{r_i \mid m'[i] \sqsubseteq m' \vee m'[i,j] \sqsubseteq m' \vee m'[j,i] \sqsubseteq m'\})$*

Verify. *The algorithm* Verify *performs the following steps:*

1. *Check, if all r_i are pair-wise distinct. If not, output 0*
2. *Use* AAggVerf *to verify τ, every $m[i]$ and the received relations which we assumed can be derived from m. In particular, all received submessages $m[i]$, appended with τ and the received r_i, τ itself, and only the submessage relations derived from m must be checked. If the validation passes, return 1, otherwise 0 resp. $\perp$ on error*

Close. *The algorithm* Close *prohibits the possibility of further redaction:*

1. *Check the validity of σ using* Verify. *If the signature is not valid, return $\perp$*
2. *If a submessage $m[k]$ is subject to redaction control, do not distribute σ_k anymore, i.e. output $\sigma' = (\sigma_c, \tau, \{\sigma_i \mid m[i] \sqsubseteq m \wedge \sigma_i \neq \sigma_k\}, \{\sigma_{i,j} \mid m[i,j] \sqsubseteq m\}, \{r_i \mid m[i] \sqsubseteq m \vee m[i,j] \sqsubseteq m \vee m[j,i] \sqsubseteq m\})$*
3. *If a submessage relation $m[k,l]$ is subject to redaction control, do not distribute $\sigma_{i,j}$ anymore, i.e. output $\sigma' = (\sigma_c, \tau, \{\sigma_i \mid m[i] \sqsubseteq m\}, \{\sigma_{i,j} \mid m[i,j] \sqsubseteq m\}, \{r_i \mid (m[i] \sqsubseteq m \vee m[i,j] \sqsubseteq m \vee m[j,i] \sqsubseteq m) \wedge \sigma_{i,j} \neq \sigma_{k,l}\})$*

The algorithms Redact and Close do not require any private keys. They only allow to remove resp. to close just a single submessage or one relation. This is done for brevity; sequentially running the given algorithms reestablishes the required and intuitive behaviour, i.e. removing a submessage along with its relations. The reason why we need to add σ_τ to the aggregate: If at least one subdocument resp. relation is closed, an adversary must be able to calculate resp. extract σ_τ, which, as we will prove in the appendix, is infeasible. Thus, the verification algorithm will not accept the signature, which reestablishes our required correctness requirement. A third party having all signatures but σ_τ can calculate it. However, this does not introduce any security problems, since the third party could give away all signatures anyway.

4.2 Instantiation Using the BGLS-Scheme

To clarify the generic description given in Sect. 4.1 we will give an instantiation now. It shows how such a scheme can be implemented using the BGLS-Scheme and it allows us to give performance measurements in Sect. 5.

Construction 2 (*RSS_2*) *For brevity, we will omit the case where already partially ordered sets are subject to signing; the algorithms can be adjusted accordingly very easily, as already shown in the high-level description of our algorithms. We will prove our generic scheme in the appendix, i.e. in App. A.*

Sign. *To sign a document $m = m[1]||\ldots||m[\ell]$, the signer signs each subdocument using the secret key sk and an additional tag τ. First, the signer signs τ, i.e. $\sigma_\tau \leftarrow (\mathcal{H}_k(\tau))^{\mathrm{sk}}$. Afterwards, ℓ pair-wise distinct nonces r_i have to be drawn uniformly. Then, each submessage $m[i] \sqsubseteq m$ is signed, i.e. $\sigma_i \leftarrow (\mathcal{H}_k(\tau||r_i||m_i))^{\mathrm{sk}}$. Note, τ must be different for each document m under sk. Afterwards, the signer calculates $\sigma_{i,j} \leftarrow (\mathcal{H}_k(\tau||r_i||r_j))^{\mathrm{sk}}$ for all $0 < i < j \leq \ell$. The tag τ is required to avoid adding subdocuments of other documents signed with the same secret key sk. Hence, τ "binds" all subdocuments to exactly one document m^τ. Afterwards, the signer aggregates all signatures σ_i and $\sigma_{i,j}$ into the final aggregated signature σ_c, i.e.:*

$$\sigma_c \leftarrow \sigma_\tau \cdot \prod_{i=1}^{\ell} \sigma_i \cdot \prod_{j=2}^{\ell} \prod_{i=1}^{i<j} \sigma_{i,j}$$

All signatures, i.e. σ_i, $\sigma_{i,j}$ and σ_τ, are sent along with the document m. Furthermore, a third party requires all random numbers r_i for verification.

Redact. *Verify σ first. To redact $m[k]$, $m[k]$ is deleted from the set of subdocuments, i.e. $m' = m \setminus m[k]$. Then the third party produces a new aggregated signature σ' over the remaining subdocuments by calculating $\sigma'_c \leftarrow \sigma_c \cdot \sigma_k^{-1}$. To redact a relation $m[i,j]$, the algorithm is similar, i.e. $m' = m \setminus m[i,j]$ and $\sigma'_c \leftarrow \sigma_c \cdot \sigma_{i,j}^{-1}$. Redacted signatures must not be further distributed. Also, all no longer required nonces must be deleted as well to maintain privacy, as shown in the generic construction.*

Verify. *To verify σ, the verifier checks whether the following equation holds:*

$$\hat{e}(\sigma_c, g) \stackrel{?}{=} \hat{e}(\mathcal{H}_k(\tau) \cdot \prod_{i=1}^{\ell} s_i \cdot \prod_{j=2}^{\ell} \prod_{i=1}^{i<j} t_{i,j}, Q)$$

where

$$s_i = \begin{cases} \mathcal{H}_k(\tau||r_i||m[i]) & \textit{if } m[i] \sqsubseteq m \\ 1 & \textit{otherwise} \end{cases} \quad \textit{and } t_{i,j} = \begin{cases} \mathcal{H}_k(\tau||r_i||r_j) & \textit{if } m[i,j] \sqsubseteq m \\ 1 & \textit{otherwise} \end{cases}$$

If so, he checks if all used r_i are pair-wise distinct. If this test is passed, the ordering and the content has been verified explicitly *and the received document is valid. The case where a given submessage resp. submessage relation is not part of the received document, does not impact on transparency, since the document could have been signed like this; A third party always knows, if a given entity does not exist, which is crucial to have a useable verification procedure, while* 1 *is the neutral element in the multiplicative group. Moreover, no information from the signature is leaked, as required by our security model. Thus, the instantiation used allows a very compact representation of the verification algorithm without introducing security flaws.*

Close. *The algorithm* Close *just no longer sends the corresponding signature. This is the same behaviour as defined in the generic construction.*

Correctness and Security of the Proposed Scheme. The proofs are relegated to App. A.

Runtime and Storage Complexity. Our construction requires $n + \frac{n(n-1)}{2} + 1 + n$ steps for signing. The dominant term is $\frac{n(n-1)}{2}$, which is in $\mathcal{O}(n^2)$. Redacting a subdocument or relation just requires two steps, namely deleting the subdocument or relation $m[k]$ and adjusting the aggregated signature to σ'. Hence, the redaction algorithm is in $\mathcal{O}(1)$. Verification is in $\mathcal{O}(n^2)$. Note, this construction requires all signatures to be available. Hence, our scheme has a storage requirement of $\mathcal{O}(n^2)$, since $n + \frac{n(n-1)}{2}$ signatures are required.

4.3 Modifications

Restricting to Sanitizer and Accountability. The proposed scheme allows public redaction. To limit redaction to explicitly denoted third parties, the signature σ_c can be altered to hold an additional signature $\sigma_2 \leftarrow \mathrm{SIGN}(sk, \mathcal{CH}(m))$, where m is the message to be signed, while $\mathcal{CH}$ is a chameleon hash [14]. The parameters of $\mathcal{CH}$ and m itself need to be delivered with σ_2. Only third parties who possess the secret key for $\mathcal{CH}$ can alter m without breaking the verification procedure. This can be enriched further to achieve third party and signer accountability [5]: $\mathcal{CH}$ could be replaced with a tag-based chameleon-hash, i.e. the one introduced by *Brzuska* et al. in [5]. Note, both types of accountability have not yet been formalized for RSS. We are confident that both formalizations are similar or even the same, though we note that this is ongoing work. The approach has already been introduced in [23] by *Pöhls* et al.

	Generation of σ in ms				Verification of σ in ms				Redaction in ms			
ℓ / Curve	10	25	50	100	10	25	50	100	10	25	50	100
128 Bit	6,350	28,675	158,557	615,546	3,675	16,638	89,233	338,156	3	10	22	32
256 Bit	39,313	170,405	667,321	2,660,354	20,323	92,828	345,360	1,401,178	9	20	44	83
384 Bit	95,555	435,902	1,740,444	6,837,645	49,203	229,935	896,825	3,580,709	15	37	71	153

Table 1. Median Runtime for the Scheme; All in ms

Binding Subdocuments and Relations. A third party can bind two or more subdocuments $m[i]$ resp. relations $m[i,j]$ to each other. In particular, it may be wanted that $m[1]$ and $m[3]$ can just be redacted together as one. This is also true for any relations; it may be wanted that a (maybe consecutive) third party is only able to redact all relations of, e.g., $m[6]$ at once. To do so, the corresponding signatures must be aggregated and distributed, e.g. in our first example $\sigma_{(1,3)} \leftarrow \sigma_1 \cdot \sigma_3$. For our second example this would be $\sigma_{(6)} \leftarrow \prod_{i=1}^{i=5} \sigma_{i,6} \cdot \prod_{i=7}^{i=\ell} \sigma_{6,i}$. Note, this does not affect the message m itself. This has already been proposed in [20]. However, we can show that for our scheme, which allows much more freedom, the signer is still able to restrict the third parties in such a sophisticated way.

5 Performance Measurements

We have implemented our scheme to demonstrate the usability despite its runtime complexity of $\mathcal{O}(n^2)$ and the fact that it is based on pairings. We used the library developed by the *National University of Maynooth*[1] [22] and the tests were run 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 a single thread to calculate the signatures; an improvement would be to parallelize signature calculations, since all but the aggregation step are independent. The source code is available upon request. We took the median of 10 runs and evaluated three sizes of curves, i.e. 128, 256 and 384 Bit. Tab. 1 shows the results for 10, 25, 50 and 100 subdocuments. As shown, for high security parameter sizes and high subdocument counts, we are considerably slower than a standard SHA-512 hash. For comparison, a SHA-512 on a document with 10 subdocuments takes 4ms and for 100 it takes 40ms. So, our implementation is at best 1,587 times slower than SHA-512 (10 subdocuments signed using a 128 bit curve). In comparison to other primitives based on pairings used in SSS our scheme can compete: A chameleon hash like *Zhang* et. al's [27] (128bit) takes 930ms to generate a single hash according to [23], while our scheme has a growth of $\mathcal{O}(n^2)$. However, all other *provably* secure and transparent schemes, i.e. [4] and [8], have the same complexity and therefore just differ by a constant factor. Hence, faster aggregate signatures would directly

[1] http://www.nuim.ie/

lead to a faster scheme. We note that for large security parameters, and a large submessage count, the scheme becomes very slow.

6 Conclusion and Open Questions

We presented a secure RSS for linear documents, that offers information-theoretical transparency and privacy. It treats content and structure as separate redactable parts; giving more freedom, which allows this RSS a wider applicability. Furthermore, we have introduced a formal and rigorous security model, which is the first to formally define the property of secure consecutive redaction control. Our scheme needs $\mathcal{O}(n^2)$ signing and verification steps. However, it allows redacting structure and content separately, which has not been possible with any scheme before, and has many applications in real world environments. Our implementation demonstrates that our construction is rather slow, but may still be useable for some real-life applications. It remains an open question, if more efficient schemes can be constructed and how we can construct an unlinkable [6] RSS with the same possibilities. Moreover, we presented an attack on the *Kundu*-Scheme [15, 16], which breaks the structural integrity protection, thus allows modifying a signed document's semantic meaning.

7 Acknowledgements

We would like to thank the anonymous reviewers for their valueable comments. Additionally, we would like to thank Christina Brzuska for some fruitful discussions concerning this paper.

References

1. Giuseppe Ateniese, Daniel H. Chou, Breno De Medeiros, and Gene Tsudik. Sanitizable signatures. In *ESORICS: Proceedings of the 10th European Symposium on Research in Computer Security*, pages 159–177. Springer-Verlag, 2005.
2. Josh Benaloh and Michael De Mare. One-way accumulators: A decentralized alternative to digital signatures. pages 274–285. Springer-Verlag, 1993.
3. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *EUROCRYPT*, pages 416–432, 2003.
4. C. Brzuska, H. Busch, O. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. 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, 2010.

5. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of Sanitizable Signatures Revisited. In *Proc. of PKC 2009*, pages 317–336. Springer, 2009.
6. Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of Sanitizable Signatures. In *Public Key Cryptography*, pages 444–461, 2010.
7. Sebastien Canard and Amandine Jambert. On extended sanitizable signature schemes. In *CT-RSA*, pages 179–194, 2010.
8. Ee-Chien Chang, Chee Liang Lim, and Jia Xu. 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, 2009. Springer-Verlag.
9. Jean-Sébastien Coron and David Naccache. Boneh et al.'s k-element aggregate extraction assumption is equivalent to the diffie-hellman assumption. In *ASIACRYPT*, pages 392–397, 2003.
10. Stuart Haber, Yasuo Hatano, Yoshinori Honda, William G. Horne, Kunihiko Miyazaki, Tomas Sander, Satoru Tezoku, and Danfeng Yao. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In *ASIACCS*, pages 353–362, 2008.
11. Tetsuya Izu, Noboru Kunihiro, Kazuo Ohta, Makoto Sano, and Masahiko Takenaka. Information security applications. chapter Sanitizable and Deletable Signature, pages 130–144. Springer-Verlag, Berlin, Heidelberg, 2009.
12. R. Johnson, D. Molnar, D. Song, and D.Wagner. Homomorphic signature schemes. In *Proceedings of the RSA Security Conference - Cryptographers Track*, pages 244–262. Springer, Feb. 2002.
13. Marek Klonowski and Anna Lauks. Extended Sanitizable Signatures. In *ICISC*, pages 343–355, 2006.
14. Hugo Krawczyk and Tal Rabin. Chameleon Hashing and Signatures. In *Symposium on Network and Distributed Systems Security*, pages 143–154, 2000.
15. A. Kundu and E. Bertino. Structural Signatures for Tree Data Structures. In *Proc. of PVLDB 2008*, New Zealand, 2008. ACM.
16. A. Kundu and E. Bertino. CERIAS Tech Report 2009-1 Leakage-Free Integrity Assurance for Tree Data Structures, 2009.
17. Ashish Kundu and Elisa Bertino. How to authenticate graphs without leaking. In *EDBT*, pages 609–620, 2010.
18. B. Liu, J. Lu, and J. Yip. XML data integrity based on concatenated hash function. *International Journal of Computer Science and Information Security*, 1(1), May 2009.
19. K. Miyazaki, S. Susaki, M. Iwamura, T. Matsumoto, R. Sasaki, and H. Yoshiura. Digital documents sanitizing problem. Technical Report ISEC2003-20, IEICE, 2003.
20. Kunihiko Miyazaki, Goichiro Hanaoka, and Hideki Imai. Digitally signed document sanitizing scheme based on bilinear maps. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, ASIACCS '06, pages 343–354, New York, NY, USA, 2006. ACM.
21. Kunihiko Miyazaki, Mitsuru Iwamura, Tsutomu Matsumoto, Ryôichi Sasaki, Hiroshi Yoshiura, Satoru Tezuka, and Hideki Imai. Digitally Signed Document Sanitizing Scheme with Disclosure Condition Control. *IEICE Transactions*, 88-A(1):239–246, 2005.
22. Louise Owens, Adam Duffy, and Tom Dowling. An Identity Based Encryption system. In *PPPJ*, pages 154–159, 2004.

23. Henrich C. Pöhls, Kai Samelin, and Joachim Posegga. 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, 2011.
24. Ravi S. Sandhu. Lattice-Based Access Control Models. *Computer*, 26:9–19, November 1993.
25. Ron Steinfeld and Laurence Bull. Content extraction signatures. In *Information Security and Cryptology - ICISC 2001: 4th International Conference*. Springer Berlin / Heidelberg, 2002.
26. Zhen-Yu Wu, Chih-Wen Hsueh, Cheng-Yu Tsai, Feipei Lai, Hung-Chang Lee, and Yufang Chung. Redactable Signatures for Signed CDA Documents. *Journal of Medical Systems*, pages 1–14, December 2010.
27. Fangguo Zhang, Reihaneh Safavi-naini, and Willy Susilo. ID-Based Chameleon Hashes from Bilinear Pairings. In *IACR Cryptology ePrint Archive*, number 208, 2003.

A Security and Correctness Proofs

Theorem 1 (The Construction is Private). *Our construction is private in the information-theoretical sense.*

Proof. Our scheme is private in the information-theoretical sense. In particular, the parts redacted are completly removed from the signature and the message. Hence, the secret bit b is perfectly hidden. The signing algorithm requires that always fresh r_i are drawn *uniformly*, while removing a random number from a uniformly distributed list leads to a uniformly distributed list again. Hence, even an unbounded adversary is not able to guess the bit better than at random. The adversary would be able to distinguish between two uniform distributions. The other way around is similar; if the redacted message would have been signed directly, while the corresponding r_i are not changed, the output is the same, prohibiting even unbounded adversaries from guessing any better than random. This implies perfect privacy. □

Theorem 2 (The Construction is Transparent and therefore private). *Our construction is transparent in the information-theoretical sense.*

Proof. Our scheme is transparent in the information-theoretical sense. In other words, the secret bit b is perfectly hidden. Our signing algorithm requires that always fresh r_i are drawn *uniformly*, while removing a random number from a uniformly distributed list leads to a uniformly distributed list again. Hence, even an unbounded adversary is not able to guess the bit better than at random. Otherwise, the adversary would be able to distinguish between two uniform distributions, which is obviously impossible. Again, the other way around is similar: If the redacted message would have been signed directly, the distributions are still uniform and it is impossible for any adversary to guess b better than at random. □

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

Proof. Note: We require that the tags τ_m are chosen unique for each message, while sk is fixed. The r_i are drawn uniformly as well. Hence, we will omit those unlikely collisions and trivial copy-attacks. Please note: We just use a single public key, which simplifies our proof. Knowing this, we can construct an adversary $\mathcal{B}$ with breaks the unforgeability of the aggregate signature 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 $\mathcal{O}^{Sign}$ and genuinely returns the answers to $\mathcal{A}$. Eventually, $\mathcal{A}$ will output a pair (m^*, σ^*). Given the transcript of the simulation, $\mathcal{B}$ checks, if (m^*, σ^*) is a trivial "forgery", i.e. a result of an allowed redaction. If so, $\mathcal{B}$ aborts the simulation and starts over. If, at some time, $\mathcal{B}$ does not need to restart, $\mathcal{B}$ outputs the tuple (m^*, σ^*) as its forgery attempt. Note that if $\exists i : \sigma^* \neq \sigma_i \wedge m_i = m^*$, then the pair (m^*, σ^*) does not win our unforgeability game and is therefore not a valid forgery attempt. This ends the simulation. We have to distinguish between two cases: (1) If $\exists i : \sigma^* = \sigma_i \wedge m_i \neq m^*$, $\mathcal{B}$ has found collision of the underlying random oracle or must have forged at least two messages. One can extract the colliding aggregates and output them as a valid forgery of the aggregate signature scheme itself. In both cases, m^* has never been queried. (2) If $\neg\exists i : \sigma^* = \sigma_i \vee m_i = m^*$. Then we have a valid forgery of a message m never queried. This breaks the unforgeability of the aggregate signature scheme.

For the BGLS-Scheme, we relegate the reader to [3] and [20], where the authors show how to break the "Diffie-Hellman-Problem" using our algorithm $\mathcal{B}$, which always outputs a valid forgery, if $\mathcal{A}$ is successful, hence with probability ϵ, if no restarts are allowed. □

Theorem 4 (The Construction is Disclosure Secure). *Our construction is disclosure secure.*

Proof. Let $\mathcal{A}$ be an algorithm winning our disclosure secure game. We can then use $\mathcal{A}$ to win our extract secure game. To do so, we let $\mathcal{B}$ use $\mathcal{A}$ as a black-box again. For every query of $\mathcal{A}$ to the oracle $\mathcal{O}^{RSign}$, $\mathcal{B}$ forwards the queries to its oracle $\mathcal{O}^{AAggSign}$. For all messages $m_c = m_i \setminus \text{MOD}_{c,i}$, $\mathcal{B}$ calls $\mathcal{O}^{Sign}$ and simulates $(m_i', \sigma_i') \leftarrow \mathsf{Close}(pk, m_i, \sigma_i, \text{MOD}_{c,i})$. Afterwards, it forwards (m_i', σ_i') genuinely to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs its forgery attempt (m^*, σ^*). If (m^*, σ^*) is non-trivial and actually winning the disclosure secure game, $\mathcal{B}$ outputs (m^*, σ^*), otherwise $\mathcal{B}$ restarts the simulation. This ends the description of our simulation. We have to distinguish between two cases:

Case 1: $\mathcal{A}$ could reconstruct parts of ADM_i

Case 2: $\mathcal{A}$ could redact parts of m_i, which were subject to disclosure control

The first case: Trivial; $\mathcal{A}$ must be able to extract signatures from the aggregate. The second case: Either $\mathcal{A}$ forged the signature or extracts subsignatures as well. To extract the subsignatures in the non-forgery case, one reverse calculates the signatures. In particular, one only needs to calculate $\sigma \backslash \sigma^*$ and output the result. The result was an aggregated signature.

In case of the BGLS-Scheme the algorithm given in [9] can use $\mathcal{B}$ to break DH, if we assume that the k-element-aggregate-extraction-assumption holds, even if the adversary has access to the signing oracle. This implies, that both cases are actually the same, if the BGLS-Scheme is utilized. $\Box$