Transparent Mergeable Redactable Signatures with Signer Commitment and Applications

Henrich C. Pöhls, Kai Samelin, Joachim Posegga and Hermann de Meer
{hp,ks,jp}@sec.uni-passau.de, demeer@uni-passau.de
Institute of IT-Security and Security-Law (ISL), University of Passau, Germany

Technical Report, Number MIP-1206
Department of Informatics and Mathematics
University of Passau, Germany
2012

Transparent Mergeable Redactable Signatures with Signer Commitment and Applications

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

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

Abstract. State-of-the-art private redactable schemes (RSS) allow the signer to un-detectably add new elements to signed data after signature generation. We introduce a RSS with a signer commitment: it prohibits any party – including a malicious signer – from adding *new* elements *after* signature generation. This protects against a malicious signer and allows using RSS for applications like timestamping. Moreover, we introduce another practically useful property: private mergeability. It allows merging two redacted versions of the same signed document into a single document with one signature. We show that neither mergeability nor signer commitment negatively impact the existing security properties. We present a provably secure redactable signature scheme that is committing, mergeable, unforgeable, private and transparent. The performance analysis of our implementation shows its practicality.

Keywords: Redactable Signatures, Privacy, Transparency, Signer Commitment, Time-Stamping, Non-Repudiation, Transparent and Private Mergeability

1 Overview

Assume *Alice* signs a set $\mathcal{S} = \{v_1, \ldots, v_\ell\}$ of elements using a secure redactable signature scheme (RSS). We use this notation of sets without loss of generality. With sets the decomposition of a message becomes easy to understand and we can facilitate mathematical notions from sets, like intersection and union to increase readability. The fundamental difference to classic signatures is that a RSS allows anyone to *redact* a subset of elements $\mathcal{R} \subseteq \mathcal{S}$, leaving a subset $\mathcal{S}' = \mathcal{S} \setminus \mathcal{R}$ along with a derived signature $\sigma_{\mathcal{S}'}$. This action is called *redaction* and can be performed by anyone. Thus, the secret signing key is not required to compute $\sigma_{\mathcal{S}'}$. A secure RSS is unforgeable comparable to classic digital signature

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

** The research leading to these results was supported by "Regionale Wettbewerbsfähigkeit und Beschäftigung", Bayern, 2007-2013 (EFRE) as part of the SECBIT project (http://www.secbit.de) and the European Community's Seventh Framework Programme through the EINS Network of Excellence (grant agreement no. [288021]).

schemes; this ensures that each element $v_i \in \mathcal{S}$ is protected against undetected subsequent modifications other than the complete removal. Hence, a positive consecutive verification of $\mathcal{S}'$ means that all elements $v_i \in \mathcal{S}'$ are authentic, i.e., the elements that remained in $\mathcal{S}'$ have not been altered and the signer remains identifiable via its public key. One application example for an RSS is the removal of personal information from signed medical records. In this application it is crucial that the redacted medical records can be used for research without an impact on the patients privacy. Next, we will describe additional existing security properties of RSS, among the above mentioned privacy.

1.1 Motivation

Overview of existing security notions. *Brzuska* et al. [5] formalized three fundamental security goals: unforgeability, privacy and transparency. Unforgeability requires that an outsider cannot forge signatures on new sets. Hence, unforgeability is comparable to the unforgeability requirements of classic digital signature schemes. Note, a valid redaction of an element from an already signed set is not considered a forgery. Furthermore, every meaningful RSS must be *private* [5]. Private means, that a third party is not be able to gain any knowledge about redacted elements from the set/signature pair without having access to them. A related notion is *transparency*; the verifier is not able to decide for a given set/signature pair if the signed set was created by signing the set or by redacting an already signed bigger set. We give formal definitions in Sect. 4.

There are also schemes that are unlinkable [7], or offer strong context-hiding [1]. In an unlinkable scheme third-parties (excluding signers) are not able to decide whether two set/signature pairs originated from the same superset; in a scheme that offers strong context hiding even the signer is not able make this decision [1,7]. However, both notions still allow signers to generate new signatures on equal or totally different sets with no proof or detection by a verifier that these correspond to a previously given signed redacted set. Hence, we motivate why we additionally need a committing behavior in many digital signature applications.

Existing unforgeable and private schemes are not committing. Our analysis reveals that some unforgeable RSSs do not protect against "forgeries" by signers. To be more precise: all current *security models* for provably secure RSSs allow the signer to add new elements to the signed set after the signature has been generated. We call schemes allowing this behavior *non-committing* RSSs. For example, *Alice* has a set $\mathcal{S}$ and signs it, generating $\sigma_{\mathcal{S}}$. *Alice* sends this to *Bob*. In a non-committing scheme, *Alice* is able to convince *Bob* at a later time that v_{new} was also an element of the larger signed set $\mathcal{S}_{\text{new}}$ with the signature $\sigma_{\mathcal{S}_{\text{new}}}$ from which one can derive by redaction the $\mathcal{S}, \sigma_{\mathcal{S}}$ that was given to *Bob*. However, v_{new} was **not** an element of $\mathcal{S}$ at the time the signature $\sigma_{\mathcal{S}}$ on $\mathcal{S}$

was generated. This gives signers the ability to update signed sets later. This an unwanted behavior for signature applications, e.g., timestamping or signing contracts.

Existing committing schemes are not private or not transparent. Some existing schemes, e.g., [24], do not allow a signer to add new elements after signature generation, but do neither fulfill the state-of-the-art privacy notion formalized by *Brzuska* et al., nor their stronger notion of transparency [5]. In applications where privacy is crucial non-private schemes cannot be used, i.e., redaction of personal identifying information from signed medical records to allow them to be used for research. Hence, they are also unsuited to protect the confidentiality of any removed state- or trade-secret.

Existing private and transparent schemes do not allow merging Another property we formally introduce in this work is mergeability. Compared to committing it is not related to existing security properties. We want to achieve that two signatures derived from the same superset can be merged into one signed set with one signature, as recently also introduced by *Lim* et al. [20]. Intuitively, merging allows a third-party to recombine two signed sets by providing a union of the not redacted elements from both sets. Only if both sets were generated by redaction from one signed superset, a third-party shall be able to compute a valid signature.

1.2 Proposed Solution: Committing and Mergeable Secure RSS

This paper shows how to construct committing, private and transparent schemes, which can be used for applications with high privacy requirements. Additionally, we show that the scheme can be mergeable.

Committing RSS. Our goal is a RSS which forces the signer to commit to a set $\mathcal{S}$, while it is still unforgeable, private, and transparent. Intuitively, we say:

> A RSS is *committing*, if a malicious signer is not able to generate a proof that a new element, which was not originally signed, was part of a given signed set. This requires that re-signing, i.e., generating a fresh signature, leads to two distinguishable signatures.

In other words, a committing scheme prohibits the signer to add new elements to an already signed set. Hence, such a RSS becomes closer to classic signature schemes and would also aid non-repudiation [14].

Note, one may argue, that this behavior does not make sense in transparent RSSs with public redaction, since the signer is always able to sign a set containing more elements and then redact a subset before giving the set to the verifier. However, a committing RSS forces the signer to know all elements to be signed *at the time of signature generation*. This can, e.g., be used for timestamping. We give more and detailed applications in Sect. 6.

Fully Private Mergeable RSS. We want to achieve that two signatures derived from the same superset can be merged into one signed set with one signature, as recently introduced by *Lim* et al. [20].

Note, the party that recombines the redacted sets by merging already knows all un-redacted elements from both subsets, and already knows that they are from the same superset.

> A RSS is *mergeable*, if third parties can merge two set/signature pairs, which have been derived from the same source, such that the resulting signature is valid over the merged set.

If mergeable, the input sets are considered *linkable*:

> A RSS is linkable, if a third party can decide for two set/signature pairs, if the two signed sets originate from the same signed superset. Moreover, if they have not been derived from the same source, nobody is able to claim so.

Trivially, mergeability together with unforgeability implies linkability, because a mergeable and unlinkable scheme must either be forgeable or linkable, a contradiction. We want to emphasize, that all security requirements, as usual, only consider the knowledge an attacker can gain from the signatures. In practical applications, the knowledge if two sets originate from the same superset could be derived from the information contained in the redacted sets.

However, being able to derive linking information from the signatures might not be wanted in applications with very strong privacy requirements. If such strict requirements are necessary, the scheme by *Ahn* et al. [1] fulfills the strictest privacy notion of “strong context hiding”. On the other hand, the scheme by *Ahn* et al. [1] cannot achieve committing behaviour due to this strict privacy, and does not offer the full flexibility to redact any $v \in \mathcal{S}$, but is limited to “quoting substrings”[3].

1.3 Our Contribution and Outline.

In this paper, we give two new concepts not yet found in secure state-of-the-art RSSs: Signer commitment and mergeability. All existing private and transparent

[3] *Ahn* et al. [1] define this as follows: “A substring of $x_1 ... x_n$ is some $x_i ... x_j$ where $i, j \in [1, n]$ and $i \leq j$. We emphasize that we are not considering subsequences. Thus, it is not possible, in this setting, to extract a signature on "I like fish" from one on "I do not like fish".” [1]

schemes are not committing, as our survey of existing RSSs in Sect. 2 shows. However, meaningful RSSs must at least be private and unforgeable [5]. Some committing schemes cannot be considered private in *Brzuska* et al.'s model [5]. The scheme introduced by *Ahn* et al. [1] does not achieve adaptive unforgeability. This paper's contribution is a secure, transparent and mergeable RSS that is committing. We prove it to be unforgeable, private, and transparent. The security properties of unforgeability, transparency and privacy are as strong as [5]. We give the first formal definition of linkability and show that it does not invade transparency, privacy or unforgeability. With transparent mergeability, we give the first formal definition of an inverse operation of redaction, which is fully privacy-preserving. Moreover we show that transparent mergeability also has no negative impact on transparency, privacy or unforgeability.

The preliminaries are given in Sect. 3, while the extended adversarial model is formally defined in a game-based manner in Sect. 4. The committing property makes our RSS closer to classic digital signature schemes. We sketch the construction in Sect. 5. Together with mergeability, the committing scheme allows for new applications. This includes time-stamping and databases, as we show in Sect. 6. The appendix contains the detailed construction (App. A), which is based on a *trapdoor-free* accumulator (App. A.1) [21,30] along with all formal proofs (App. B).

2 State of the Art

The general concept of RSSs has been introduced as "content extraction signatures" by *Steinfeld* et al. [31] and in the same year by *Johnson* et al. as "homomorphic signatures" [15]. Their ideas have been extended to work on tree-structured data [5,16,28] and on arbitrary graphs [17]. A related concept are sanitizable signature schemes (SSS) [2,6], where the sanitizer does not redact elements. Instead, it can change elements to arbitrary strings $\in \{0,1\}^*$. SSSs require sanitizers to know a secret and do not allow public alterations. Their aim is different, so they are not discussed in this paper.

In the following, the related work is structured by security properties, see Tab. 1 for an overview of the offered security properties and the analytical runtime overhead for each scheme.

Privacy and Transparency. Many proposed schemes are not private in *Brzuska* et al.'s security model [5], i.e., a third party can see that at least one element has been redacted. Exemplarily, see [12,15,24,31]. Also many non-transparent RSSs can be attacked using the missing transparency as a "side-channel" for information to break privacy, i.e., if the position of a redacted element is visible, one is able to gain information about the original document that one should not be

Scheme	Type	Unforgeable	Transparent	Private	Context-Hiding	Linkable	Mergeable	Committing.	Runtime
[1]	Quoting	○[a]	✓	✓	✓	✗	✗	○[b]	$\mathcal{O}(n \cdot log(n))$
[5]	Trees	✓	✓	✓	✗	✓	✓	✗	$\mathcal{O}(n^2)$
[9]	Lists	✓	✓	✓	✗	✓	✗	○[c]	$\mathcal{O}(n^2)$
[16]	Trees	✗	✗	✗	✗	✓	✓	✗	$\mathcal{O}(n)$
[18]	Trees	✓	✓	✓	✗	✓	✓	✗	$\mathcal{O}(n^2)$
[23]	Sets	✓	✓	✓	✗	✓	✓	✗	$\mathcal{O}(n)$
[24]	Lists	✓	✗	✗	✗	✓	✓	✓	$\mathcal{O}(n)$
[29]	Lists	✓	✓	✓	✗	✓	✓	✗	$\mathcal{O}(n^2)$
This	Sets	✓	✓	✓	✗	✓	✓	✓	$\mathcal{O}(n)$

Table 1. Comparison of some RSSs; n is the number of redactable elements

[a] They achieve selective unforgeability.

[b] The strong context-hiding property renders a discussion obsolete.

[c] Depending on the underlying primitives.

able to gain from a private scheme. As privacy preserving applications require an RSS to be private, these schemes cannot be used.

Committing. *Brzuska* et al. name the possibility that the signer is allowed to add new elements after the initial signature generation "dynamic updates" [5]. This has been derived from *Kundu* and *Bertino*, who also allow the same concept under this name [16]. Examples for updateable RSSs can be found in [5,16,23,29]. We named the schemes following this behavior "*non-committing* RSS". The strong context-hiding property of the scheme introduced by *Ahn* et al. [1] discourages any discussions about dynamic updates, since even the signer is only able to generate unlinkable signatures. In a non-committing RSSs, a third party is not able to distinguish between elements which were contained in the set from the beginning or ones which have been added by the signer after the signing procedure.

Linkability and Mergeability. In the field of RSSs, all existing provably *secure* and *transparent* constructions consider only how to redact elements. The opposite –reinstate previously redacted elements– in a controlled way has neither been formalized nor considered yet for **transparent** schemes [20]. The closest works are [26] and [20]. Using *Merkle*-hash-trees the construction from [26] allows a secret-controlled reconstructability, but it is not private. Notions of mergeability are also given originally by *Merkle* for hash-trees [22], but they are not private as well. Similarly, the construction offered in [20] is not private.

Pöhls et al. [27] than left it as an open question if it possible to construct a scheme which is private and mergeable. This paper gives a positive answer to all of these questions. The ability to sign a set, redact all elements, distribute the signed empty set and then later release elements that can be reconstructed resembles some requirements of blind signatures [10]. However, we do not require the unlinkability or untraceability offered by blind signatures in our use case. Let us emphasize, that *Ahn* et al.'s "strong context hiding" [1] explicitly forbids reconstruction of data.

3 Preliminaries

We extend the framework given in [5] with definitions for Merge and Link.

Definition 1 (Transparent, Mergeable and Committing RSS_C). *A mergeable and committing RSS_C consists of six efficient algorithms. In particular, $\mathsf{RSS}_C := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact}, \mathsf{Link}, \mathsf{Merge})$ 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* λ *is the security parameter*

Sign. *The algorithm Sign gets as input the secret key* sk *and the set* $\mathcal{S}$*. It outputs* $(\mathcal{S}, \sigma_\mathcal{S}) \leftarrow \mathsf{Sign}(\mathrm{sk}, \mathcal{S})$

Verify. *The algorithm Verify outputs a bit* $d \in \{0, 1\}$ *indicating the correctness of the signature* $\sigma_\mathcal{S}$*, w.r.t.* pk*, protecting* $\mathcal{S}$*.* 1 *stands for a valid signature. In particular:* $d \leftarrow \mathsf{Verify}(\mathrm{pk}, \mathcal{S}, \sigma_\mathcal{S})$

Redact. *The deterministic[4] algorithm Redact takes as input the set* $\mathcal{S}$*, the public key* pk *of the signer, a valid signature* $\sigma_\mathcal{S}$ *and a set* $\mathcal{R}$ *of elements to be redacted. The algorithm outputs* $(\mathcal{S}', \sigma'_\mathcal{S}) \leftarrow \mathsf{Redact}(\mathrm{pk}, \mathcal{S}, \sigma_\mathcal{S}, \mathcal{R})$*, where* $\mathcal{S}' = \mathcal{S} \backslash \mathcal{R}$*.* $\mathcal{R}$ *is allowed to be the empty set or contain more than one element. We denote the transitive closure of* $\mathcal{S}$*, w.r.t. to Redact and* $\sigma_\mathcal{S}$*, as* $span_\models(\mathcal{S}, \sigma_\mathcal{S})$*, following [9].* $span_\models(\mathcal{S}, \sigma_\mathcal{S})$ *contains all set/signature pairs derivable from* $(\mathcal{S}, \sigma_\mathcal{S})$*:* $span_\models(\mathcal{S}, \sigma_\mathcal{S}) = \{(\mathcal{U}, \sigma_U) \leftarrow \mathsf{Redact}(\mathrm{pk}, \mathcal{S}, \sigma_\mathcal{S}, \mathcal{R}_i) \mid \mathcal{R}_i \in \mathcal{P}(\mathcal{S})\}$*, where* $\mathcal{P}(\mathcal{S})$ *denotes the power set of* $\mathcal{S}$*. We want to emphasize that* $\emptyset \in \mathcal{P}(\mathcal{S})$*. On error, the algorithm outputs* $\perp$

Link. *The algorithm Link takes as input a set/signature pair* $(\mathcal{S}, \sigma_\mathcal{S})$*, a public key* pk *and a second set/signature pair* $(\mathcal{T}, \sigma_\mathcal{T})$*. It outputs a bit* $d \in \{0, 1\}$*, indicating, if* $(\mathcal{S}, \sigma_\mathcal{S})$ *and* $(\mathcal{T}, \sigma_\mathcal{T})$ *have both been derived from the same valid set/signature pair* $(\mathcal{U}, \sigma_\mathcal{U})$ *by Redact or Merge. It outputs* 1*, if* $(\mathcal{S}, \sigma_\mathcal{S}) \in span_\models(\mathcal{U}, \sigma_\mathcal{U}) \wedge (\mathcal{T}, \sigma_\mathcal{T}) \in span_\models(\mathcal{U}, \sigma_\mathcal{U})$ *and* 0 *otherwise. Note, for Link neither* $\mathcal{U}$*,* $\sigma_\mathcal{U}$ *nor* $span_\models(\mathcal{U}, \sigma_\mathcal{U})$ *are required.*

Merge. *The algorithm Merge takes as input the public key* pk *of the signer, two sets* $\mathcal{S}$ *and* $\mathcal{V}$*, and the corresponding signatures* $\sigma_\mathcal{S}$ *and* $\sigma_\mathcal{V}$*. Note, we require* $\sigma_\mathcal{S}$ *and* $\sigma_\mathcal{V}$ *to be valid on* $\mathcal{S}$ *and* $\mathcal{V}$*. It outputs* $(\mathcal{U}, \sigma_\mathcal{U}) \leftarrow \mathsf{Merge}(\mathrm{pk}, \mathcal{S}, \sigma_\mathcal{S}, \mathcal{V}, \sigma_\mathcal{V})$*, where* $\mathcal{U} = \mathcal{S} \cup \mathcal{V}$ *and* $\sigma_\mathcal{U}$ *is valid on* $\mathcal{U}$*. On error, the algorithm outputs* $\perp$

[4] This algorithm may be probabilistic, but a deterministic definition allows for a more intuitive description.

We require that one can efficiently identify all the elements $v_i \in \mathcal{S}$ from a given $\mathcal{S}$. Redact is a public operation, as common in RSSs. Thus, all parties can redact the set, which includes the signer, as well as any intermediate recipient. We require the correctness properties to hold, i.e., every genuinely signed set must verify with overwhelming probability. The same must hold for any set/signature pair generated using Redact or Merge, i.e., using Redact or Merge on valid set/signature pairs must result in a new set that again verifies with overwhelming probability. Last, we require Link outputs the correct bit with overwhelming probability as well. We give formal definitions of the correctness requirements in an extended version of this paper.

4 Security Model

This section introduces the security model. We use the security model introduced by *Brzuska* et al. in [5] as our starting point. In particular, we also require unforgeability, privacy and transparency. We extend and adapt the model to cover committing, linkability, as well as mergeability. Additionally, we introduce the properties of *merge transparency* and *merge privacy*, not considered in [20]. As usual, the following definitions only address the information a third party can derive from the signature $\sigma_{\mathcal{S}}$ alone; e.g., if in a real application the redactions that took place are obvious, it may be trivial for attackers to detect them.

Unforgeability. No one should be able to produce a valid signature on a set $\mathcal{S}^*$ verifying under pk with elements outside the transitive closure of any set $\mathcal{S}$ received, without having access to the corresponding secret key sk. That is, even if an attacker can adaptively request signatures on different documents, it remains impossible to forge a signature for a new set not queried. This is analogous to the standard unforgeability requirement for other signature schemes. We say that a RSS is **unforgeable**, if for every probabilistic polynomial time (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 1 returns 1, is negligible (as a function of λ).

Privacy. The verifier should not be able to gain any knowledge about redacted elements without having access to them. The adversary can choose two tuples $(\mathcal{S}_0, \mathcal{R}_0)$ and $(\mathcal{S}_1, \mathcal{R}_1)$. A redaction of $\mathcal{R}_0$ from $\mathcal{S}_0$ is required to result in the same set as redacting $\mathcal{R}_1$ from $\mathcal{S}_1$. The two sets are input to a "Left-or-Right" oracle which signs and redacts $(\mathcal{S}_b, \mathcal{R}_b)$. The adversary wins, if it can decide which pair was used to as input to create the oracle's output. This is similar to the standard indistinguishability notion for encryption schemes. We say that a RSS for documents is **private**, if for PPT adversary $\mathcal{A}$ the probability that the game depicted in Fig. 2 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ).

```
**Experiment** $\mathsf{Unforgeability}^{\mathsf{RSS}_C}_{\mathcal{A}}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $(\mathcal{S}^*, \sigma^*_{\mathcal{S}}) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pk)$
    let $i = 1, 2, \ldots, q$ index adaptive queries
    so $(\mathcal{S}^i, \sigma^i_{\mathcal{S}})$ are answers of the oracle
  return 1, if
    $\mathsf{Verify}(pk, \mathcal{S}^*, \sigma^*_{\mathcal{S}}) = 1$ and
    $\forall i, 1 \leq i \leq q : \mathcal{S}^* \not\subseteq \mathcal{S}^i$
```

Fig. 1. Game for Unforgeability

```
**Experiment** $\mathsf{Privacy}^{\mathsf{RSS}_C}_{\mathcal{A}}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $b \xleftarrow{\$} \{0, 1\}$
  $d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot),\mathsf{LoRRedact}(\ldots,sk,b)}(pk)$
    where oracle $\mathsf{LoRRedact}$
    for input $\mathcal{S}_0, \mathcal{S}_1, \mathcal{R}_0, \mathcal{R}_1$:
    if $\mathcal{S}_0 \setminus \mathcal{R}_0 \neq \mathcal{S}_1 \setminus \mathcal{R}_1$, return $\perp$
    $(\mathcal{S}, \sigma_{\mathcal{S}}) \leftarrow \mathsf{Sign}(sk, \mathcal{S}_b)$
    return $(\mathcal{S}', \sigma'_{\mathcal{S}}) \leftarrow \mathsf{Redact}(pk, \mathcal{S}, \sigma_{\mathcal{S}}, \mathcal{R}_b)$.
  return 1, if $b = d$
```

Fig. 2. Game for Privacy

```
**Experiment** $\mathsf{Transparency}^{\mathsf{RSS}_C}_{\mathcal{A}}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $b \xleftarrow{\$} \{0, 1\}$
  $d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot),\mathsf{Sign/Redact}(\ldots,sk,b)}(pk)$
    where oracle $\mathsf{Sign/Redact}$
    for input $\mathcal{S}, \mathcal{R}$:
      if $\mathcal{R} \not\subseteq \mathcal{S}$, return $\perp$
      if $b = 0$:  $(\mathcal{S}, \sigma_{\mathcal{S}}) \leftarrow \mathsf{Sign}(sk, \mathcal{S})$,
                   $(\mathcal{S}', \sigma'_{\mathcal{S}}) \leftarrow \mathsf{Redact}(pk, \mathcal{S}, \sigma_{\mathcal{S}}, \mathcal{R})$
      if $b = 1$:  $\mathcal{S}' \leftarrow \mathcal{S} \setminus \mathcal{R}$
                   $(\mathcal{S}', \sigma'_{\mathcal{S}}) \leftarrow \mathsf{Sign}(sk, \mathcal{S}')$,
    finally return $(\mathcal{S}', \sigma'_{\mathcal{S}})$.
  return 1, if $b = d$
```

Fig. 3. Game for Transparency

```
**Experiment** $\mathsf{Committing}^{\mathsf{RSS}_C}_{\mathcal{A}}(\lambda)$
  $(pk^*, sk^*) \leftarrow \mathcal{A}(1^\lambda)$
  $\mathcal{V}^* = \{v^*_1, \ldots, v^*_k \mid v^*_i \leftarrow \mathcal{A}(pk^*, sk^*)\}$
  $(\mathcal{V}^*, \sigma^*_{\mathcal{V}}) \leftarrow \mathsf{Sign}(sk^*, \mathcal{V}^*)$
  $(\mathcal{S}^*, \sigma^*_{\mathcal{S}}) \leftarrow \mathcal{A}((pk^*, sk^*), (\mathcal{V}^*, \sigma^*_{\mathcal{V}}))$
  return 1, if
    $\mathsf{Verify}(pk^*, \mathcal{S}^*, \sigma^*_{\mathcal{S}}) = 1$ and
    $\mathsf{Link}(pk^*, \mathcal{S}^*, \sigma^*_{\mathcal{S}}, \mathcal{V}^*, \sigma^*_{\mathcal{V}}) = 1$ and
    $\mathrm{span}_{\models}(\mathcal{S}^*, \sigma^*_{\mathcal{S}}) \supsetneq \mathrm{span}_{\models}(\mathcal{V}^*, \sigma^*_{\mathcal{V}})$
```

Fig. 4. Game for Signer Commitment

Transparency. The verifier should not be able to decide whether a signature has been created by the signer directly, or through the redaction algorithm Redact. The adversary can choose one tuple $(\mathcal{S}, \mathcal{R})$. The pair is input for a "Sign/Redact" oracle which signs and redacts ($b = 0$) or redacts and then signs ($b = 1$). The adversary wins, if it can decide which way was taken. We say that a RSS is **transparent**, if for every PPT adversary $\mathcal{A}$, the probability that the game depicted in Fig. 3 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ).

Committing. No party should be able to add *new* elements to a signed set. Especially, even a malicious signer is not able to generate a proof that a new element, which was not originally signed, was part of a given signed set. Thus, the adversary wins, if it can add new elements to a signed set, it can even choose the key pair and all elements contained. Re-signing is not possible due to Link. We say that a RSS is **committing**, if for every PPT adversary $\mathcal{A}$ the probability that the game depicted in Fig. 4 returns 1, is negligible (as a function of λ).

Linkability. Nobody should be able to deny that two set/signatures pairs originate from the same source, if they do. Vice versa, if two set/signature pairs do

```
**Experiment** $\mathsf{Linkability}_{\mathcal{A}}^{\mathsf{RSS}_C}(\lambda)$
  $(pk^*, sk^*) \leftarrow \mathcal{A}(1^\lambda)$
  $(\mathcal{S}^*, \sigma_{\mathcal{S}}^*, \mathcal{T}^*, \sigma_{\mathcal{T}}^*) \leftarrow \mathcal{A}(pk^*, sk^*)$
  return 1, if
      $\mathsf{Verify}(pk^*, \mathcal{S}^*, \sigma_{\mathcal{S}}^*) = 1$ and
      $\mathsf{Verify}(pk^*, \mathcal{T}^*, \sigma_{\mathcal{T}}^*) = 1$ and
      $\mathsf{Link}(pk^*, \mathcal{S}^*, \sigma_{\mathcal{S}}^*, \mathcal{T}^*, \sigma_{\mathcal{T}}^*) = 0$ and
      $\mathrm{span}_{\models}(\mathcal{S}, \sigma_{\mathcal{S}}) \subseteq \mathrm{span}_{\models}(\mathcal{T}, \sigma_{\mathcal{T}})$
          **or**
      $\mathsf{Link}(pk, \mathcal{S}^*, \sigma_{\mathcal{S}}^*, \mathcal{T}^*, \sigma_{\mathcal{T}}^*) = 1$ and
      $\mathrm{span}_{\models}(\mathcal{S}, \sigma_{\mathcal{S}}) \nsubseteq \mathrm{span}_{\models}(\mathcal{T}, \sigma_{\mathcal{T}})$
```

Fig. 5. Game for Linkability

```
**Experiment** $\mathsf{Mergeability}_{\mathcal{A}}^{\mathsf{RSS}_C}(\lambda)$
  $(pk^*, sk^*) \leftarrow \mathcal{A}(1^\lambda)$
  $(\mathcal{S}^*, \sigma_{\mathcal{S}}^*, \mathcal{T}^*, \sigma_{\mathcal{T}}^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(pk)$
  return 1, if
      $\mathsf{Verify}(pk^*, \mathcal{S}^*, \sigma_{\mathcal{S}}^*) = 1$ and
      $\mathsf{Verify}(pk^*, \mathcal{T}^*, \sigma_{\mathcal{T}}^*) = 1$ and
      $\mathsf{Link}(pk^*, \mathcal{S}^*, \sigma_{\mathcal{S}}^*, \mathcal{T}^*, \sigma_{\mathcal{T}}^*) = 1$ and
      $\mathsf{Merge}(pk^*, \mathcal{S}^*, \sigma_{\mathcal{S}}^*, \mathcal{T}^*, \sigma_{\mathcal{T}}^*) = \perp$
```

Fig. 6. Game for Mergeability

not originate from the same source nobody should be able to claim that they do. Hence, the adversary has two options to win: (1) The attacker outputs two signature/message pairs which are derived from the same signature/message pair, but where Link outputs 0 or, (2) the attacker outputs two signature/message pairs which are not derived from the same signature/message pair, but where Link outputs 1. Thus, we say that a RSS is **linkable**, if for a PPT adversary $\mathcal{A}$, the probability that the game depicted in Fig. 5 returns 1, is negligible (as a function of λ).

Mergeability. Nobody should be able to prohibit a reconstruction by Merge, if the sets are linkable. In this game, the adversary wins, if it can output two signature/message pairs, which are linkable but not mergeable, i.e., Merge returns $\perp$. We say that a RSS is **mergeable**, if for every PPT adversary $\mathcal{A}$ the probability that the game depicted in Fig. 6 returns 1, is negligible (as a function of λ).

Merge Privacy. If a merged set is given to another third party, the party should not be able to derive any information besides what is contained in the merged set, i.e., a verifier should not be able to decide which elements have been merged from what set. In the game, the adversary wins, if it can decide which elements were redacted and merged afterwards. This is encapsulated within the LoRRedact oracle. We say that a scheme RSS is **merge private**, if for every PPT adversary $\mathcal{A}$, the probability that the game depicted in Fig. 7 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ).

Merge Transparency. If a merged set is given to another third party, the party should not be able to decide whether the set has been created by Sign or through Merge. The adversary can choose one tuple $(\mathcal{S}, \mathcal{R})$. This pair is input to a "Sign/Redact" oracle which returns the original pair $(b = 0)$ or redacts and then merges $(b = 1)$. The adversary wins, if it can decide which way was taken.

```
$\textbf{Experiment}$ Merge Privacy$_{\mathcal{A}}^{\mathsf{RSS}_C}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $b \xleftarrow{\$} \{0,1\}$
  $d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot),\mathsf{LoRRedact}(\ldots,sk,b)}(pk)$
      where oracle $\mathsf{LoRRedact}$
      for input $\mathcal{S}, \mathcal{R}_0, \mathcal{R}_1$:
      $(\mathcal{S}, \sigma_\mathcal{S}) \leftarrow \mathsf{Sign}(sk, \mathcal{S})$
      if $\mathcal{R}_0 \nsubseteq \mathcal{S} \vee \mathcal{R}_1 \nsubseteq \mathcal{S}$: return $\perp$
      if $b = 0$: $(\mathcal{S}', \sigma'_\mathcal{S}) \leftarrow \mathsf{Redact}(pk, \mathcal{S}, \sigma_\mathcal{S}, \mathcal{R}_0)$
          $(\mathcal{R}', \sigma'_\mathcal{R}) \leftarrow \mathsf{Redact}(pk, \mathcal{S}, \sigma_\mathcal{S}, \mathcal{S} \setminus \mathcal{R}_0)$
      if $b = 1$: $(\mathcal{S}', \sigma'_\mathcal{S}) \leftarrow \mathsf{Redact}(pk, \mathcal{S}, \sigma_\mathcal{S}, \mathcal{R}_1)$
          $(\mathcal{R}', \sigma'_\mathcal{R}) \leftarrow \mathsf{Redact}(pk, \mathcal{S}, \sigma_\mathcal{S}, \mathcal{S} \setminus \mathcal{R}_1)$
      return $\mathsf{Merge}(pk, \mathcal{S}', \sigma'_\mathcal{S}, \mathcal{R}', \sigma'_\mathcal{R})$
  return 1, if $b = d$
```

Fig. 7. Game for Merge Privacy

```
$\textbf{Experiment}$ Merge Transparency$_{\mathcal{A}}^{\mathsf{RSS}_C}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $b \xleftarrow{\$} \{0,1\}$
  $d \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot),\mathsf{Sign/Redact}(\ldots,sk,b)}(pk)$
      where oracle $\mathsf{Sign/Redact}$ for input $\mathcal{S}, \mathcal{R}$:
          if $\mathcal{R} \nsubseteq \mathcal{S}$, return $\perp$
          $(\mathcal{S}, \sigma_\mathcal{S}) \leftarrow \mathsf{Sign}(sk, \mathcal{S})$
          if $b = 0$:   $(\mathcal{S}', \sigma'_\mathcal{S}) \leftarrow (\mathcal{S}, \sigma_\mathcal{S})$,
          if $b = 1$:   $(\mathcal{T}', \sigma'_\mathcal{T}) \leftarrow \mathsf{Redact}(pk, \mathcal{S}, \sigma_\mathcal{S}, \mathcal{R})$
                        $(\mathcal{R}', \sigma'_\mathcal{R}) \leftarrow \mathsf{Redact}(pk, \mathcal{S}, \sigma_\mathcal{S}, \mathcal{S} \setminus \mathcal{R})$
                        $(\mathcal{S}', \sigma'_\mathcal{S}) \leftarrow \mathsf{Merge}(pk, \mathcal{T}', \sigma'_\mathcal{T}, \mathcal{R}', \sigma'_\mathcal{R})$
      finally return $(\mathcal{S}', \sigma'_\mathcal{S})$.
  return 1, if $b = d$
```

Fig. 8. Game for Merge Transparency

We say that a scheme RSS is **merge transparent**, if for every PPT adversary $\mathcal{A}$, the probability that the game depicted in Fig. 8 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ). We emphasize that the notions of merge transparency and merge privacy are very similar to the notions of privacy and transparency, as they achieve comparable goals. We call a construction secure, if it is unforgeable, (merge) transparent and (merge) private. Here, we see mergeability as a feature and not as a security requirement.

5 Construction

In a nutshell, our generic construction is based upon an accumulator ACC, which must be collision-resistant without trusted setup and indistinguishable. Both properties are formally defined in the appendix. The signer uses ACC to accumulate all elements v_i of $\mathcal{S}$ into one accumulator value a. For each element v_i that is accumulated a witness p_i is calculated. Finally, the accumulator value a is signed with a standard signature scheme $\mathit{\Pi}$, e.g., RSA. From these underlying building blocks our construction also inherits the security model.

The four basic ideas are:
(1) the accumulator value a remains unchanged after a redaction has taken place, hence all derived subset/signature pairs are linkable by a.
(2) Redactions are private, as without knowledge of the corresponding witness p_i (and v_i) nobody can verify if v_i was "in" the signed a.
(3) Mergeability is achieved, as knowledge of an element/witness pair allows a third party to add it back into a signed set.
Finally, (4) unforgeability comes from signing a.

The construction is generic, and proved secure if the accumulator and the signature scheme fulfill the security requirements. We give the complete algorithmic description in Appendix A and security proofs in Appendix B.

6 Applications

This new property enables the use of RSS in new domains. In this paper, we discuss the application of RSSs for privacy preserving timestamping, partial but authentic disclosure from archives, verifiable transparent database unions of data in the cloud. State-of-the-art security models did not achieve the required properties for these applications.

Privacy Preserving Timestamping. Timestamps are issued by a third party and produce evidence that can be used to show that the timestamped data existed at that particular time and was not modified since. A secure timestamping service (TSS) [13] gets a hash of the data and generates a timestamp. The data remains confidential, as the TSS gets only a hash of the data. Assume that *Alice* wants to produce evidence to *Charly* that a set $\mathcal{S}$ of documents existed in that exact form. To do so, *Alice* first produces a *proof of existence* (PoE) as follows: *Alice* signs the set of documents $\mathcal{S}$, with a committing yet mergeable RSS. *Alice* then redacts the set completely, i.e, $\mathcal{S}' = \emptyset$. Then *Alice* submits the signature $\sigma_{\mathcal{S}}$ and the empty set to the secure timestamping service TSS, which timestamps the PoE. Later in time, *Alice* can send the complete $\mathcal{S}$ or any subset and the timestamped signature $\sigma_{\mathcal{S}}$ to *Charly.* Having the signature and TSS's timestamp, *Charly* is convinced that the elements he sees in $\mathcal{S}$ existed at the time of timestamping. This allows *Alice* to disclose any subset $\mathcal{S}' \subseteq \mathcal{S}$ without the need to timestamp each subset in advance. Still, *Alice*'s privacy is protected towards both entities: TSS does not see the elements in $\mathcal{S}$ that it timestamped. And with our transparent and private RSS, *Charly* is not able to see if he only received a subset. Existing solutions, which also only need one timestamp, leak the latter information to *Charly.*

Partial, but authentic, disclosure of archived contracts. Electronic documents like contracts or financial records are stored in archives. Due to legal compliance the archiving process needs to ensure the integrity, authenticity and timeliness of the data archived. In particular, the archival of documents must result in a verifiable proof that the documents have existed at the time of entering the archive and have not changed since. Hence, businesses have such archives digitally signed by third parties, such as financial auditors. If legal duties, or other circumstances, require to share some, but not all, documents from the archive using redactable signatures allow protecting trade secrets, by removing them, while it keeps the auditors signature as evidence. Imagine a company is given a court order to provide all their archived information relating to a certain business transaction as evidence in court, i.e., during an e-discovery [11]. The company is only in need to present the documents containing only the information relevant for the court case. Companies neither need nor want to release the full signed archive nor documents with trade-secrets unnecessary to disclose.

Keeping the signature of the auditor valid helps dissipating doubt about providing falsified information up front. A naïve solution requires each document-part being signed individually and creates an overhead in storage and management of the set of signatures.

Verifiable and transparent database unions of data distributed in the cloud. Distributed storage of data base records is another application area. Reasons to split database records are manifold; e.g., for efficiency large databases can be be split and distributed over several database servers. For reasons of data protection, data must be split into personal parts and parts that contain data, which on its own is anonymous. For example, data protection is important for bio data banks, which are collections of samples of human bodily substances (e.g., blood or DNA) that are or can be associated with persona data. Each single field of each record in a complete database is signed using a redactable and mergeable signature scheme, which allows records or databases to be split by redaction. To clarify, assume a DB with three records $\{A, B, C\}$ is split in three parts with just one element each: Part 1 is created by redaction of $\{B, C\}$ from the complete set, part 2 by redaction of $\{A, C\}$, and part 3 by redacting $\{B, C\}$. All parts retain a valid signature. Clients can request records from a server and check their integrity by verification of the signature. This allows separating the records such that highly confidential values are not stored on servers with an insufficient security clearance. Mergeability allows combining database records later. If single records are split it allows each single database server to combine several records in one answer. The recipient can verify their signature. Moreover, the client can ask several database servers if he has the needed clearance and merge their answers and gain assurance that they come from the same source and establish that the records have not been modified. Due to the privacy of the scheme, parts that have been removed during splitting cannot be reconstructed without having access to them. Hence, parts without a need for a high-level of protection can be stored on servers with lower security requirements. Additionally, a third-party can carry out the merging step. Due to merge transparency, even the information that an answer has been created through a merge remains unknown to the recipient.

7 Conclusion

We have introduced the notion of committing redactable signature schemes. This security property prohibits a signer to add new elements to a signed set after signature generation. Moreover, we have formalized the notion of mergeability, the inverse of redactions. Combined, these properties allow using redactable signature schemes in more application scenarios, e.g., timestamping and distributed databases. This has not been possible before. We have analyzed state-of-the-art redactable signature schemes and found that those that offer the same level of privacy and transparency, are not committing and those that show committing or mergeable behavior are neither private nor transparent. Our committing and mergeable scheme is provably secure, that is, it is unforgeable, (merge) transparent and (merge) private. We achieve the same security regarding transparency, which implies privacy, and unforgeability as the more recent state-of-the-art schemes. In particular our security is comparable to that of *Brzuska* et al. [5]. Our construction is based on a trapdoor-free accumulator and an unforgeable signature scheme. Theoretical runtime complexity for the signature generation is $\mathcal{O}(n)$. Verification, redaction and merging are also in $\mathcal{O}(n)$. Our implementation's performance suggests, that our scheme is reasonable fast.

References

1. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. Computing on authenticated data. Cryptology ePrint Archive, Report 2011/096, 2011. `http://eprint.iacr.org/`.
2. G. Ateniese, D. H. Chou, B. De Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS: Proceedings of the 10th European Symposium on Research in Computer Security*, pages 159–177. Springer-Verlag, 2005.
3. N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, pages 480–494, 1997.
4. J. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In *EUROCRYPT*, pages 274–285, 1993.
5. 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 *ACNS*, pages 87–104. Springer, 2010.
6. 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.
7. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schroeder. Unlinkability of sanitizable signatures. In *Public Key Cryptography*, pages 444–461, 2010.
8. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, pages 61–76, 2002.
9. E.-C. Chang, C. L. Lim, and J. Xu. Short Redactable Signatures Using Random Trees. In *CT-RSA*, pages 133–147, Berlin, Heidelberg, 2009. Springer-Verlag.
10. D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.
11. EU Article 29 Data Protection Working Party. Document 1/2009 on pre-trial discovery for cross border civil litigation. 00339/09/EN, WP 158, Feb. 2009.
12. S. Haber, Y. Hatano, Y. Honda, W. G. Horne, K. Miyazaki, T. Sander, S. Tezoku, and D. Yao. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In *ASIACCS*, pages 353–362, 2008.
13. S. Haber and W. S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3:99–111, 1991.
14. ISO/IEC 13888-1. Information technology - surity techniques - non-repudiation. part 1: General. ISO/IEC, 1997.

15. 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.
16. A. Kundu and E. Bertino. Structural Signatures for Tree Data Structures. In *Proc. of PVLDB 2008*, New Zealand, 2008. ACM.
17. A. Kundu and E. Bertino. How to authenticate graphs without leaking. In *EDBT*, pages 609–620, 2010.
18. Ashish Kundu, Mikhail J. Atallah, and Elisa Bertino. Leakage-free redactable signatures. In *CODASPY*, pages 307–316, 2012.
19. J. Li, N. Li, and R. Xue. Universal accumulators with efficient nonmembership proofs. In *ACNS*, pages 253–269, 2007.
20. S. Lim, E. Lee, and C.-M. Park. A short redactable signature scheme using pairing. *Security and Communication Networks*, 5(5):523–534, 2012.
21. H. Lipmaa. Secure accumulators from euclidean rings without trusted setup. In *ACNS*, LNCS, pages 224–240. Springer, 2012.
22. R.C Merkle. A certified digital signature. In *Advances in Cryptology*, 1989.
23. K. Miyazaki, G. Hanaoka, and H. Imai. Digitally signed document sanitizing scheme based on bilinear maps. In *Proc. of ASIACCS'06*, ASIACCS '06, pages 343–354, New York, USA, 2006. ACM.
24. K. Miyazaki, M. Iwamura, T. Matsumoto, R. Sasaki, H. Yoshiura, S. Tezuka, and H. Imai. Digitally Signed Document Sanitizing Scheme with Disclosure Condition Control. *IEICE Transactions*, 88-A(1):239–246, 2005.
25. K. Nyberg. Fast accumulated hashing. In *FSE*, pages 83–87, 1996.
26. H. C. Pöhls. Verifiable and revocable expression of consent to processing of aggregated personal data. In *ICICS 2008*, LNCS 5308, pages 279–293. Springer, 2008.
27. H. C. Pöhls, A. Bilzhause, K. Samelin, and J. Posegga. Sanitizable signed privacy preferences for social networks. In *DICCDI*, LNI. GI, Oct. 2011.
28. K. Samelin, H. C. Pöhls, A. Bilzhause, J. Posegga, and H. de Meer. On Structural Signatures for Tree Data Structures. In *ACNS, 10th International Conference*, volume 7341 of *LNCS*, pages 171–187. Springer-Verlag, 2012.
29. K. Samelin, H. C. Pöhls, A. Bilzhause, J. Posegga, and H. de Meer. Redactable signatures for independent removal of structure and content. In *ISPEC*, volume 7232 of *LNCS*, pages 17–33. Springer-Verlag, 2012.
30. T. Sander. Efficient accumulators without trapdoor extended abstracts. In *ICICS*, pages 252–262, 1999.
31. R. Steinfeld and L. Bull. Content extraction signatures. In *ICISC 2001: 4th International Conference*. Springer Berlin / Heidelberg, 2002.

A Construction of a Committing, Mergeable and Transparent and Unforgeable RSS

This section gives a quick outline of the construction before we continue with an algorithmic explanation. Our construction is based upon an accumulator ACC, which must be (1) trapdoor-free[5] and (2) not leak information about additional members, i.e., the digests must be indistinguishable. The second building block is

[5] Or the trapdoor must not be known to the signer

an unforgeable signature scheme (UNF-CMA). The construction is generic, and secure if the two building blocks fulfill the security requirements. From these underlying building blocks our construction also inherits the security model. Our prototype uses the ideas given in [30]. *Lipmaa*'s construction [21] is another candidate.

A.1 Accumulator: Trapdoor-Freeness and Indistinguishability

Cryptographic accumulators (ACC) have been introduced by *Benaloh* and *de Mare* [4]. They hash a potentially very large set into a short single value a, called the accumulator. To later prove that a given value v_i was actually used to calculate a, a witness $p_i \in \mathcal{P}_{pk}$ is calculated. $\mathcal{Y}_{pk}$ denotes the input domain, while the output domain is denoted as $\mathcal{X}_{pk}$. $pk \in \mathcal{K}$ expresses the public parameters required by ACC, whereas $\mathcal{K}$ denotes the key space. We require that ACC is trapdoor-free [30], while an adversary cannot decide how many elements are contained in a given accumulator without having access to **all** of them. We do not require any dynamic updates [8], non-membership witnesses [19], non-deniability [21], or quasi-commutativity [25] for our scheme to work. This allows a wider range of accumulators to be used as the underlying building block for our construction.. The following algorithmic description is derived from [21].

Definition 2 (Cryptographic Accumulators). *A cryptographic accumulator ACC consists of five efficient (PPT) algorithms. In particular, ACC := (Setup, Gen, Eval, Wit, Ver) such that:*

Setup. *The algorithm Setup is the parameter generator. On input of the security parameter λ and the **public** random tape ω, it outputs the system parameter parm, i.e., parm $\leftarrow$ Setup$(1^\lambda, \omega)$*

Gen. *The algorithm Gen is the instance generator. On input of the security parameter λ and parm, it outputs* pk*:* pk $\leftarrow$ *Gen*$(1^\lambda, \mathit{parm})$

Dig. *The algorithm Dig takes as input the set $\mathcal{S}$ to accumulate, the public parameters* pk *and outputs an accumulator value a, i.e, $a \leftarrow$ Dig*$(\mathrm{pk}, \mathcal{S})$

Proof. *The algorithm Proof takes as input the public parameters* pk*, a value $y \in \mathcal{Y}_{\mathrm{pk}}$ and returns a witness p from a witness space $\mathcal{P}_{\mathrm{pk}}$, if $y \in \mathcal{S}$ was input to Dig, i.e., Dig$(\mathrm{pk}, \mathcal{S})$, and $\perp$ otherwise. Hence, it outputs $p \leftarrow$ Proof$(\mathrm{pk}, y, \mathcal{S})$*

Verf. *The verification algorithm Verf takes as input the public key* pk*, an accumulator $a \in \mathcal{X}_{\mathrm{pk}}$, a witness p, and a value $y \in \mathcal{Y}_{\mathrm{pk}}$ and outputs a bit $d \in \{0, 1\}$ indicating whether p is a valid proof that y has been accumulated into a. Hence, it outputs $d \leftarrow$ Verf$(\mathit{parm}, \mathrm{pk}, a, y, p)$*

We require the correctness properties to hold. Refer to [3] for a formal definition.

```
**Experiment** $\mathsf{Collision-Resistance}^{\mathsf{ACC}}_{\mathcal{A}}(\lambda, \omega)$
    $\mathsf{parm} \leftarrow \mathsf{Setup}(1^\lambda, \omega)$
    $pk \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{parm})$
    $(m^*, S^*, p^*) \leftarrow \mathcal{A}(\omega, \mathsf{parm}, pk)$
    $a \leftarrow \mathsf{Dig}(pk, \mathcal{S}^*)$
    return 1, if
        $\mathsf{Ver}(pk, a, m^*, p^*) = 1$ and
        $m^* \notin \mathcal{S}^*$
```

Fig. 9. Game for Collision-Resistance Without Trusted Setup

```
**Experiment** $\mathsf{Indistinguishable}^{\mathsf{ACC}}_{\mathcal{A}}(\lambda, \omega)$
    $\mathsf{parm} \leftarrow \mathsf{Setup}(1^\lambda, \omega)$
    $pk \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{parm})$
    $b \overset{\$}{\leftarrow} \{0, 1\}$
    $d^* \leftarrow \mathcal{A}^{\mathsf{LoRHash}(\ldots, b, pk)}(\omega, \mathsf{parm}, pk)$
        where oracle $\mathsf{LoRHash}$ for input $\mathcal{S}, \mathcal{R}$:
        $z \overset{\$}{\leftarrow} \mathcal{Y}_{pk}$
        if $b = 1$: return $(\mathsf{Dig}(pk, \mathcal{S} \cup \mathcal{R} \cup z),$
        $\{(y_i, p_i) \mid p_i \leftarrow \mathsf{Proof}(pk, y_i \in \mathcal{S}, \mathcal{S} \cup \mathcal{R} \cup z)\})$
        if $b = 0$: return $(\mathsf{Dig}(pk, \mathcal{S} \cup z),$
        $\{(y_i, p_i) \mid p_i \leftarrow \mathsf{Proof}(pk, y_i \in \mathcal{S}, \mathcal{S} \cup z)\})$
    return 1, if $d = b$
```

Fig. 10. Game for Indistinguishability

Collision-Resistant Without Trusted Setup. An adversary should not be able find a valid witness/element pair (p^*, y^*) for a given accumulator a, even if it is allowed to see the random tape ω used for generation of parm. This is related to the CRS-model. The honest generation of pk can be verified by using verifiable use of randomness. Refer to [21] for a detailed explanation. Another way, not requiring a stronger accumulator, is to let a third party, a.k.a. the accumulator manager, handle the public parameter generation. It depends on the concrete use case what method is more appropriate. In our case, we only require the signer not to know any trapdoor information that allows finding collisions in a trivial way. We use the trapdoor-free approach. We call a family of accumulators **collision-resistant without trusted setup**, if the probability that the experiment depicted in Fig. 9 returns 1, is negligible (as a function of λ). In particular, the adversary has to find a witness/element pair which verifies, while the element has not been accumulated.

Indistinguishable Membership Count. We also require that an adversary who does **not** know **all** elements in $\mathcal{S}$ is not able to decide how many additional elements the accumulator a contains. This behavior is crucial to maintain transparency for our construction. In particular, we add a nonce as a blinding value to prohibit attackers from re-calculating the accumulator a.

We say that an accumulator is indistinguishable, if the probability that the game depicted in Fig. 10 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ). Here, the adversary can choose the input, and has to guess, if the digest has only one more member ($b = 0$) or more ($b = 1$). The blinding value z is chosen by the oracle at random. The basic idea is to require that one additionally accumulated blinding value is enough to hide that an accumulator has additional members. Please note that the witnesses are also returned.

A.2 Our Construction

Construction 1 (Committing and Mergeable RSS.) *A non-updateable and reconstructable* RSS *consists of six efficient algorithms. In particular,* $\mathcal{RSS}_C :=$ (KeyGen, Sign, Verify, Redact, Link, Merge) *such that:*

KeyGen. *The algorithm* KeyGen *generates the key pair in the following way:*

1. *Generate a key pair of standard signature scheme* $\Pi = (\mathsf{SKeyGen}, \mathsf{SSign}, \mathsf{SVerify})$ *like RSA, i.e.,* $(\mathrm{pk}_s, \mathrm{sk}_s) \leftarrow \Pi.\mathsf{SKeyGen}(\lambda)$
2. *Generate or get the public parameters required for the accumulator, i.e., run* $\mathsf{parm} \leftarrow \mathsf{Setup}(1^\lambda, \omega)$ *and* $\mathrm{pk}_h \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{parm})$
3. *Output* $((\mathrm{pk}_s, \omega, \mathsf{parm}, \mathrm{pk}_h), (\mathrm{sk}_s))$

Sign. *To sign a set* $\mathcal{S}$*, perform the following steps:*

1. *Draw a random value:* $y \overset{\$}{\leftarrow} \mathcal{Y}_{\mathrm{pk}}$
2. *Accumulate all elements of* $\mathcal{S}$ *and* y*:* $a \leftarrow \mathsf{Dig}(\mathrm{pk}, y \cup \mathcal{S})$
3. $\mathcal{P} = \{p_i \mid \mathsf{Proof}(\mathrm{pk}, y_i, \mathcal{S})\}$
4. *Sign* a*:* $\mathsf{SSign}(\mathrm{sk}_s, a)$
5. *Output* $(\mathcal{S}, \sigma_{\mathcal{S}})$*, where* $\sigma_{\mathcal{S}} = (\mathrm{pk}_s, \sigma_a, \{(y_i, p_i) \mid y_i \in \mathcal{S} \wedge p_i \in \mathcal{P}\})$

Verify. *To verify a signature* $\sigma_{\mathcal{S}} = (\mathrm{pk}_s, \sigma_a, \{(y_i, p_i) \mid y_i \in \mathcal{S} \wedge p_i \in \mathcal{P}\})$*, perform:*

1. *Verify* σ_a*, w.r.t.* pk_s *and* a*, using* $\mathsf{SVerify}$
2. *Check, if* pk_h *and* parm *have been generated honestly, using* ω
3. *For each element* $y_i \in \mathcal{S}$ *using its witness check, if* $\mathsf{Verf}(\mathrm{pk}, a, y_i, p_i) = 1$

Redact. *To redact a subset* $\mathcal{R} \subseteq \mathcal{S}$*, the algorithm performs the following steps:*

1. *Check the validity of* $\sigma_{\mathcal{S}}$ *using* Verify*. If* $\sigma_{\mathcal{S}}$ *is not valid, return* $\perp$
2. *Output* $(\mathcal{S}', \sigma'_{\mathcal{S}})$*, where* $\sigma'_{\mathcal{S}} = (\mathrm{pk}_s, \sigma_a, \{(y_i, p_i) \mid y_i \in \mathcal{S} \setminus \mathcal{R}\})$
   Note, all unnecessary p_j *for* $y_j \in \mathcal{R}$ *are no longer distributed.*

Link. *The link algorithm performs the following steps:*

1. *Verify* $\sigma_{\mathcal{S}}$ *and* $\sigma_{\mathcal{T}}$*. If any signature is not valid, return* $\perp$
2. *Return* 1*, if the values* $a_{\mathcal{S}}$ *and* $a_{\mathcal{T}}$ *protected by the inner signatures* $\sigma_{a,\mathcal{S}}$ *and* $\sigma_{a,\mathcal{T}}$*are equal,* 0 *otherwise*

Merge. *To merge two set/signature pairs* $(\mathcal{S}, \sigma_{\mathcal{S}})$ *and* $(\mathcal{T}, \sigma_{\mathcal{T}})$*, perform:*

1. *Check, if* $\mathsf{Link}(\mathrm{pk}, \mathcal{S}, \sigma_{\mathcal{S}}, \mathcal{T}, \sigma_{\mathcal{T}}) = 1$*, if not, return* $\perp$
2. *Output* $(\mathcal{S} \cup \mathcal{T}, \sigma_{\mathcal{U}})$*, where* $\sigma_{\mathcal{U}} = ((\mathrm{pk}_s, \sigma_a, \{(y_i, p_i) \mid y_i \in \mathcal{S} \cup \mathcal{T}\})$

B Security Proofs

Theorem 1 (The Construction is Correct and Secure). *Our construction is secure and correct, if the underlying signature scheme is unforgeable and the underlying accumulator is trapdoor-free and indistinguishable. We call a redactable signature scheme secure if it is unforgeable, (merge) transparent and (merge) private.*

We give proofs for unforgeability, transparency, which implies privacy. After this, we prove the new properties of committing and the notions of merge privacy and merge transparency. We sometimes have to remain sketchy due to limited space.

Theorem 2 (The Construction is Unforgeable). *Our construction is unforgeable, if the underlying signature scheme is unforgeable and the underlying accumulator collision-resistant without trusted setup.*

Proof. The only case we have to consider to win the game for unforgeability given in Fig. 1, is that $S^* \nsubseteq S$. We denote the adversary winning the unforgeability game as $\mathcal{A}_{unf}$. We can then derive, that the forgery must fall into at least one of the following categories:

1. The value protected by the underlying signature σ_a^* has never been queried.
2. The value protected by the underlying signature σ_a^* has been queried, but $S^* \nsubseteq S$, i.e., the documents protected are not in the transitive closure of any queried signature. This case has to be divided further:

   2a. $S \nsubseteq S^*$ 2b. $S^* \supset S$

$S^* \subseteq S$, is not a forgery, but a redaction. We show that each case leads to a contradiction about the security of the underlying primitives.

Case 1. In this case, we can use $\mathcal{A}_{unf}$ as a black-box to break the unforgeability of the underlying signature algorithm. In particular, we can construct the algorithm $\mathcal{A}_{unfSS}$ in the following way:

1. Forward pk of the SS to forge to $\mathcal{A}_{unf}$
2. Any queries to the signing oracle from $\mathcal{A}_{unf}$ are forwarded to $\mathcal{A}_{unfSS}$'s own signing oracle and genuinely returned
3. Eventually, $\mathcal{A}_{unf}$ outputs a pair $(S^*, \sigma_{\mathcal{S}}^*)$
4. $\mathcal{A}_{unfSS}$ returns (a^*, σ_a^*), which can be extracted from $\sigma_{\mathcal{S}}^*$
5. If the digest a has been digested, abort

As we have required that a^* has never been signed, the tuple (a^*, σ_a^*) breaks the standard unforgeability requirement of the underlying signature scheme.

Case 2a. In this case, an element has been added to the set, which has not been signed by the signing oracle. This can be derived from the requirements that $S^* \not\subseteq S$ and $S \not\subseteq S^*$ must yield. Hence, we break the collision-resistance of the underlying accumulator by letting $\mathcal{A}_{col}$ use $\mathcal{A}_{unf}$ as a black-box:

1. Generate a key-pair of a SS. Forward $\mathcal{H}$ and pk to $\mathcal{A}_{unf}$
2. Any queries to the signing oracle from $\mathcal{A}_{unf}$ are genuinely answered by $\mathcal{A}_{col}$
3. Eventually, $\mathcal{A}_{unf}$ outputs a pair $(S^*, \sigma^*_{\mathcal{S}})$
4. $\mathcal{A}_{col}$ returns $(a, \{(v_i, p_i) \mid v_i \in \mathcal{S}^*\})$, which are contained in $\sigma^*_{\mathcal{S}}$ and $\mathcal{S}^*$
5. If every element has been queried, abort.

As there exists an element $v^*_i \in \mathcal{S}^*$ which has not been accumulated, $\mathcal{A}_{col}$ breaks the collision-resistance of the underlying accumulator by outputting (a, v^*_i, p^*_i)

Case 2b. The same as case 2a, since at least one element must have been added.

Theorem 3 (The Construction is Transparent). *Our construction is transparent and therefore private, if the output of the accumulator is indistinguishable.*

Proof. Let $\mathcal{A}_{tra}$ be an adversary winning the transparency game as defined in Fig. 3. We can then use $\mathcal{A}_{tra}$ as a black-box in an algorithm $\mathcal{A}_{indis}$ to show that the underlying accumulator is not indistinguishable.

1. Receive the public key of the hash function
2. Generate a key pair for a signature scheme to simulate the signing oracle
3. Forward the public keys to $\mathcal{A}_{tra}$
4. Simulate the LoRRedact-Oracle by forwarding $\mathcal{R}$, $\mathcal{S}$ and pk to the LoRHash-Oracle. Sign the digest using the signature scheme's secret key.
5. Eventually, $\mathcal{A}_{tra}$ outputs its guess b^*

$\mathcal{A}_{indis}$ outputs b^* as its own guess. The success probability of $\mathcal{A}_{indis}$ is exactly the same as $\mathcal{A}_{tra}$, as the elements are passed through.

Theorem 4 (The Construction is committing). *Our construction is committing, if the accumulator is collision-resistant without trusted setup.*

Proof. Let $\mathcal{A}_{updateable}$ be an adversary winning the committing game as defined in Fig. 4. We can then use $\mathcal{A}_{updateable}$ as a black-box in an algorithm $\mathcal{A}_{col}$ which breaks the collision-resistance without trusted setup of the accumulator.

1. Receive the key-pair (pk^*, sk^*) from $\mathcal{A}_{updateable}$
2. Receive the set $\mathcal{V}^*$ from $\mathcal{A}_{updateable}$ and sign it
3. Forward $(\mathcal{V}^*, \sigma^*_{\mathcal{V}}, pk^*, sk^*)$ to $\mathcal{A}_{updateable}$

4. Eventually, $\mathcal{A}_{updateable}$ outputs a pair $(S^*, \sigma_{\mathcal{S}}^*)$
5. $\mathcal{A}_{col}$ returns $(a, \{(v_i, p_i) \mid v_i \in \mathcal{S}^*\})$, which are contained in $\sigma_{\mathcal{S}}^*$ and $\mathcal{S}^*$
6. Abort, if the colliding tuples are trivial, i.e., no new element is contained.

As there exists an element $v_i^* \in \mathcal{S}^*$ which has not been accumulated, $\mathcal{A}_{col}$ breaks the collision-resistance without trusted setup by outputting (a, v_i^*, p_i^*).

Theorem 5 (Our Construction is Mergeable and Linkable). *Our construction is mergeable and linkable, if the accumulator is collision-resistant without trusted setup and the signature scheme is unforgeable.*

Proof. (Sketch) Each signed set has a unique and constant accumulator value a. Valid redactions do not change this signed value. Witnesses of previously redacted elements stay valid members of their set's a. It is linkable unless the attacker is able to change a outside of a valid redaction. To achieve this the attacker would need to forge the signature or it implies an adversary against the collision-resistance of the underlying accumulator. Constructing a simulator along with an extractor is straight forward.

Theorem 6 (Our Construction is Merge Private and Merge Transparent). *Our construction is merge private and merge transparent.*

Proof. The contents and values of merged and freshly signed signatures are *equal*. This implies, that our construction is *information-theoretically* merge private and *information-theoretically* merge transparent.

C Complexity Analysis and Performance Evaluation

For signing, our scheme requires $\mathcal{O}(n)$ steps, where n is the size of the set. We chose an accumulator with trusted setup, and we exclude the modulus generation from the measurements, as they are only done once for the setup. This requires $\mathcal{O}(n)$ steps for *each* witness generated. Hence, our overall runtime complexity is $\mathcal{O}(n^2)$. Using a more sophisticated tree-based evaluation algorithm, the complexity of witness generation can be improved to $\mathcal{O}(log^2(n) \cdot n)$ [3]. The accumulator being central means a faster accumulator in $\mathcal{O}(n)$ leads to a total runtime complexity of $\mathcal{O}(n)$. Verifying a signature is in $\mathcal{O}(n)$. Consequently, redacting, linking and merging are also in $\mathcal{O}(n)$, since the only relevant step is verifying the given signature(s). The storage complexity is also $\mathcal{O}(n)$, as for each element $v_i \in \mathcal{S}$, a witness p_i exists. We use an *Intel* Q9550 Quad Core @2.83 GHz with 3 GiB of RAM and RSA as the signature algorithm. For the accumulator we implemented [3]. Utilized is *Windows XP* (32 Bit) with *Java* `1.7.0_01-b08`. We measured with different modulus sizes of 512, 1024 and 2048 Bit. Excluded from timing: (1) RSA key pair generation, since it is reasonable

	Sign			Verification			Linking			Merging		
ℓ Modulus	10	50	100	10	50	100	10	50	100	10	50	100
512 Bit	1.4	15.5	55.8	0.3	1.5	3.9	0.6	3.0	7.8	0.7	3.0	7.8
1024 Bit	14.1	161.5	464.7	3.8	27.1	46.0	6.8	53.5	91.3	6.8	53.5	91.2
2048 Bit	24.8	403.1	1,440.3	4.8	32.3	56.6	8.6	63.4	112.3	8.6	63.4	112.9

Table 2. Median Runtime; in s

to assume that a signer already owns a key pair. The time spent for the key pair generation becomes negligible for large document counts. (2) generation of suitable RSA modulus for the accumulator, as it depends on the actual use case how to generate it. Shown is the median of `10` runs. All intermediate results are stored in RAM to avoid any impact of disk access. The results can be seen in Tab. 2. As shown, our construction is useable, but becomes slow for a large elements count. However, if a faster secure accumulator becomes available, the runtime will decrease, rendering our scheme useable even for large numbers of elements. We provide the source code on request.