On Updatable Redactable Signatures

Henrich C. Pöhls[1,2,⋆], Kai Samelin[⋆⋆]

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

Abstract. Redactable signatures allow removing parts from signed documents. State-of-the-art security models do not capture the possibility that the signer can "update" signatures, i.e., add new elements. Neglecting this, third parties can generate forgeries. Moreover, there are constructions which permit creating a signature by merging two redacted messages, if they stem from the same original. Our adjusted definition captures both possibilities. We present a provably secure construction in the standard model, which makes use of a novel trapdoor-accumulator.

1 Introduction

Assume we sign a set $\mathcal{S} = \{v_1, v_2, \ldots, v_\ell\}$, generating a signature σ protecting $\mathcal{S}$.[1] The use of a redactable signature scheme ($\mathcal{RSS}$) now allows removing elements from $\mathcal{S}$: a verifying signature σ' for a subset $\mathcal{S}' \subseteq \mathcal{S}$ can be derived by anyone. This action is called a *redaction*. For this, no secret key is not required, i.e., redacting is a public operation. This possibility is contrary to standard digital signatures, which do not permit any alterations. Public redactions are especially useful, if the original signer is not reachable anymore, e.g., in case of death, or if it produces too much overhead to resign a message every time an alteration is necessary, e.g., if communication is too costly. Hence, $\mathcal{RSS}$s partially address the "digital document sanitization problem" [36]. Formally, $\mathcal{RSS}$s are a proper subset of ($\mathcal{P}$-)homomorphic signatures [1]. The obvious applications for $\mathcal{RSS}$s are privacy-preserving handling of medical records, the removal of the date-of-birth from certificates from job applications, and the removal of identifying information for age-restricted locations from XML-files or the cloud [7,27,30,40,41,42,43,45]. Real implementations are given in [40,44,46]. However, existing provably secure constructions offer the possibility of "dynamic updates". In a nutshell, dynamic updates allow the signer to add new elements to existing signatures. This captures the ideas given in [8,29]. Hence, a signer can add new elements without the need to re-sign everything, e.g., to add new

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

[⋆⋆] This work was partly supported by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE and by the Hessian LOEWE excellence initiative within CASED, while working at CASED.

[1] [14,19,43] show how to treat more complex data-structures with an $\mathcal{RSS}$ for sets.

I. Boureanu, P. Owesarski, and S. Vaudenay (Eds.): ACNS 2014, LNCS 8479, pp. 457–475, 2014.
© Springer International Publishing Switzerland 2014

credentials or new database entries. This is useful in many situations, e.g., if it is too costly to re-sign a database completely for each new entry added. In other words, updates may become less expensive than a complete re-sign. Unfortunately, this also allows forgeries according to the existing security models whenever the *adversary* is given access to an "update-oracle". Jumping ahead, this oracle essentially models the adversary's possibility to ask the signer to adaptively "update" signatures with elements of its own choice. Given the aforementioned application scenarios, this possibility must not be neglected. Related, and present in many $\mathcal{RSS}$s in the literature is the possibility of "merging": given two set/signature pairs derived from the same signature, one can combine them into a single *merged*, set/signature pair $(\mathcal{S}, \sigma)$.

Example. Consider bio data banks, which are collections of samples of human bodily substances (e.g., DNA) that are associated with personal data. While the personal data part might be quite small, the medical data is often very large. However, from the association with personal data follows the necessity to obey strict data protection requirements. Once this association is removed, the protection can be (partly) forgo and the handling of this data becomes simpler. Hence, one wants to store tables containing columns with personally identifying information separate from those that contain other data, which on its own is anonymous. Splitting the data allows to store it on servers with different security requirements, and possibly different costs. Also, the outsourcing party wants to allow the database service provider to repartition the database as it sees suitable for an optimal operation. The use of a private, updatable and mergeable $\mathcal{RSS}$ now allows protecting the integrity by signing each single field of each row in each table as an element. In $\mathcal{RSS}$s, privacy says that no information about redacted elements can be derived from signatures, *if one has no access to them.* In our example, this means that the low-security servers gain *no* information about the high-security columns. A formal definition is given in Sect. 2. To clarify, consider a simple database with one table and three columns (`id`, `Name`, `DNA`). When this database is signed, we get one signature over the complete database. Now, we can split the complete record into one table containing only (`id, Name`) pairs, and a second one holding (`id, DNA`) pairs. To do so, we generate two separate signatures by using the redact operation of the $\mathcal{RSS}$. Now, the $\mathcal{RSS}$ can further be used by each database server to generate and hand valid signature over to clients for any elements returned by their queries. If a query only returns one row, the database server redacts all other rows and sends the resulting signature to the client. The client is able to verify that the result set is from the original signer and the returned values have not been modified. In turn, mergeability now allows deriving a *single* valid signature from formerly split database records. Hence, the database service provider can undo the splitting by merging the result set, and the signatures, before returning it. A property we define in Sect. 2, named *merge transparency*, even hides the information that a signature has been created through a merge operation from the verifying client.

If the splitting was done for privacy, any client or third-party service which is allowed to query for private data, e.g., (`id, Name`), and (`id, DNA`) can merge the database server's answers and gain assurance that the merged result containing data with three columns (`id, Name, DNA`) comes from the trusted source and establish that the records have not been modified. On the other hand, dynamic updates have been motivated in previous work before and allow the signer, i.e., the database owner, to dynamically add new records to the database.

In our scenario, we require that: (1) an attacker cannot generate non-legitimate signatures, (2) redacted information remains hidden, and (3) that all algorithms create *linkable* versions of the same document. The third requirement makes $\mathcal{RSS}$s useable in practice: an attacker must not be able to generate "clones" of a signed set by gradually redacting different elements. As an example, assume that `Name` consists of several columns (`FirstName`, `MiddleName`, `FamilyName`). Then, we do not want that the DNA record of "Rose" "Fitzgerald" "Kennedy" can be duplicated without detection into several sets by redaction of some of the elements, e.g., pretending two signed DNA records exist: one for a member of the "Kennedy" family and another one for a woman with the first name of "Rose". Merging thus serves as a test when its questionable whether or not two records are derived from the same original: once two signed sets are available, their signatures only merge, if they are linkable.

State-of-the-Art. The concept of $\mathcal{RSS}$s has been introduced as "content extraction signatures" by *Steinfeld* et al. [45], and, in the same year, as "homomorphic signatures" [28] by *Johnson* et al. A security model for sets has been given by *Miyazaki* et al. [37]. Their ideas have been extended to work on trees [14,30,42], on lists [19,43], and also on arbitrary graphs [30]. There are also some schemes which offer *context-hiding*, a very strong privacy notion, and variations thereof, e.g., [1,3,4]. Context-hiding schemes allow to generate signatures on derived sets with no possibility to detect whether these correspond to an already signed set in a statistical sense. To some extend, the concept of sanitizable signature schemes ($\mathcal{SSS}$) is related [2,13,15,16,17,21,31]. In an $\mathcal{SSS}$, the sanitizer does not redact elements, but can change "admissible elements" to arbitrary strings, i.e., $v_i' \in \{0,1\}^*$. $\mathcal{SSS}$s require sanitizers to know a secret and therefore do not allow public alterations. Even though they seem to be related, the aims and security models of $\mathcal{SSS}$s and $\mathcal{RSS}$s substantially differ on a detailed level [22]. Hence, $\mathcal{SSS}$s are not discussed in any more detail in this paper. Most recent advances generalize similar concepts. These are normally referred to as "($\mathcal{P}$-)homomorphic signatures", "functional signatures", and/or "delegateable signatures". Noteworthy work includes [1,3,4,5,10,12,20,23]. In this paper, we focus on $\mathcal{RSS}$s, while our results are applicable to the aforementioned primitives.

In the field of $\mathcal{RSS}$s, all existing provably *private* constructions only consider how to redact elements. The opposite — reinstating previously redacted elements, i.e., merging signatures — in a controlled way has neither been formalized nor have security models been properly discussed. Notions of mergeability are initially given by *Merkle* for hash-trees [35], but these are not private in the context of $\mathcal{RSS}$s. The closest existing works mentioning merging in our context are [28,33,38,39]. However, neither of the mentioned schemes is fully private in our model, while [28] is even forgeable — merging from any signed set is possible.

Contribution. As aforementioned, current security models do not correctly capture the possibility that some signatures can be updated, i.e., that the signer can freely add new elements. Additionally, they also do not discuss that signatures can, under certain circumstances, be merged. In this paper, we introduce an extended security model explicitly capturing both possibilities. Our contribution is therefore manifold: (1) We present some shortcomings in existing security models, which do not consider the case of updating signatures by the signer. We show how an adversary can construct forgeries, if this possibility is neglected. (2) We propose a countermeasure: we augment the state-of-the-art security model with explicit access to an "update-oracle", which an adversary can query adaptively. We also rigorously define the notions of "update privacy" and "update transparency". Jumping ahead, both properties describe which information can be derived from an updated signature. (3) We introduce a formal definition of "mergeability", i.e., under which circumstances signatures can be merged into a single one. With private and transparent mergeability, we give the first security model of the inverse operation of redaction, extending the work done in [33]. Again, both properties aim to formalize which information an adversary can obtain from a merged signature. We prove that merging signatures has no negative impact on existing security properties. (4) We show how the new and old notions are related to each other, extending the work by *Brzuska* et al. [14]. (5) We derive a provably secure construction, meeting our enhanced definitions. (6) For our construction, we deploy trapdoor-accumulators. This construction is of independent interest. Moreover, it turns out that we do not require any kind of standard signature scheme, which is a very surprising result on its own. Also, our construction proves that the statement given in [30] that accumulators are not sufficient for $\mathcal{RSS}$s is not true.

2 Preliminaries and Security Model

We heavily modify the security model introduced by *Brzuska* et al. [14], as we explicitly allow merging and updating signatures. We do so by introducing the algorithms Merge[2] and Update.

Definition 1 (Mergeable and Updatable $\mathcal{RSS}$). *A mergeable and updatable $\mathcal{RSS}$ consists of six efficient algorithms. Let $\mathcal{RSS} := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact}, \mathsf{Update}, \mathsf{Merge})$, 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* $(\mathcal{S}, \sigma, \tau) \leftarrow \mathsf{Sign}(1^\lambda, \mathrm{sk}, \mathcal{S})$*. Here,* τ *is a tag*

[2] Merge was named "combine" in [33].

Verify. *The algorithm* Verify *outputs a bit* $d \in \{0,1\}$ *indicating the correctness of the signature* σ*, w.r.t.* pk *and* τ*, protecting* $\mathcal{S}$*.* 1 *stands for a valid signature, while* 0 *indicates the opposite. In particular:* $d \leftarrow \mathsf{Verify}(1^\lambda, \mathrm{pk}, \mathcal{S}, \sigma, \tau)$

Redact. *The algorithm* Redact *takes as input a set* $\mathcal{S}$*, the public key* pk *of the signer, a tag* τ*, and a valid signature* σ *and a set* $\mathcal{R} \subset \mathcal{S}$ *of elements to be redacted. The algorithm outputs* $(\mathcal{S}', \sigma', \tau) \leftarrow \mathsf{Redact}(1^\lambda, \mathrm{pk}, \mathcal{S}, \sigma, \mathcal{R}, \tau)$*, where* $\mathcal{S}' = \mathcal{S} \setminus \mathcal{R}$*.* $\mathcal{R}$ *is allowed to be* $\emptyset$*. On error, the algorithm outputs* $\perp$

Update. *The algorithm* Update *takes as input a verifying set/signature/tag tuple* $(\mathcal{S}, \sigma, \tau)$*, the secret key* sk *and a second set* $\mathcal{U}$*. It outputs* $(\mathcal{S}', \sigma', \tau) \leftarrow \mathsf{Update}(1^\lambda, \mathrm{sk}, \mathcal{S}, \sigma, \mathcal{U}, \tau)$*, where* $\mathcal{S}' = \mathcal{S} \cup \mathcal{U}$*, and* σ' *is a verifying signature on* $\mathcal{S}'$*. On error, the algorithm outputs* $\perp$

Merge. *The algorithm* Merge *takes as input the public key* pk *of the signer, two sets* $\mathcal{S}$ *and* $\mathcal{V}$*, a tag* τ*, and the corresponding signatures* $\sigma_\mathcal{S}$ *and* $\sigma_\mathcal{V}$*. We require that* $\sigma_\mathcal{S}$ *and* $\sigma_\mathcal{V}$ *are valid on* $\mathcal{S}$ *and* $\mathcal{V}$*. It outputs the merged set/signature/tag tuple* $(\mathcal{U}, \sigma_\mathcal{U}, \tau) \leftarrow \mathsf{Merge}(1^\lambda, \mathrm{pk}, \mathcal{S}, \sigma_\mathcal{S}, \mathcal{V}, \sigma_\mathcal{V}, \tau)$*, where* $\mathcal{U} = \mathcal{S} \cup \mathcal{V}$ *and* $\sigma_\mathcal{U}$ *is valid on* $\mathcal{U}$*. On error, the algorithm outputs* $\perp$

We assume that one can efficiently, and uniquely, identify all the elements $v_i \in \mathcal{S}$ from a given set $\mathcal{S}$. All algorithms, except Sign and Update, are public operations, as common in $\mathcal{RSS}$s. In other words, all parties can redact and merge sets, which includes the signer, as well as any intermediate recipient. The correctness properties must also hold, i.e., every genuinely signed, redacted, merged, or updated set must verify. The same is true for updates and merging signatures. This must even hold transitively, i.e., the history of the signature must not matter. τ does not change on any operation. As we allow merging signatures, unlinkability cannot be achieved: τ makes signatures linkable.

On the Security Implications of Dynamic Updates. In our model, Update and Merge are explicitly defined, while in existing work they are present only *implicitly*. The schemes we want to review have in common that they allow dynamic updates and merging of signatures. In particular, there is a very subtle possibility, undermining existing schemes' unforgeability in current security models. Even without explicit algorithms, the following succeeds regardless of a scheme's implementation.

For the following, all signatures share the same tag τ. An adversary $\mathcal{A}$ can break the state-of-the-art unforgeability [14] of an $\mathcal{RSS}$ in the following way: $\mathcal{A}$ queries its signing oracle with a set $\{A\}$, receiving a signature σ_A. Afterward, $\mathcal{A}$ requests the signer to update $\{A\}$ to $\{A, B\}$, receiving a signature $\sigma_{A,B}$. Additionally, $\mathcal{A}$ requests a second update of $\{A\}$ to $\{A, C\}$, receiving $\sigma_{A,C}$. $\mathcal{A}$ can then "merge" $(\{A, B\}, \sigma_{A,B})$ and $(\{A, C\}, \sigma_{A,C})$ to a new verifying signature $(\{A, B, C\}, \sigma^*)$. This set/signature pair is considered a forgery in existing models, while this may be a wanted behavior, e.g., if in medical records new diseases are appended by two different medical doctors. Let us stress that we have introduced a new oracle, namely the update-oracle. However, even if the adversary cannot request updates of his own choosing, it can merge them into a forgery, according to existing models, once the signer performs two different updates on a signed set. Hence, either dynamic updates must be completely prohibited, or the existing security model must be altered. As our application scenario proves, dynamic updates have their merits and enable many practical applications. Therefore, we chose to take the second path. Note, that our example is tailored for sets, while some schemes address lists and trees. However, the aforementioned possibility is not limited to sets, but works on all the schemes aforementioned with minor adjustments. We stress that we explicitly took care that most current existing schemes, e.g., [14,37,42,43], can be considered secure in our enhanced security model. An exception are schemes which offer context-hiding (and their variants). This property discourages any discussions about dynamic updates, as an updated signature cannot be linked against an "old" one. This allows meeting the property of (statistical) unlinkability: derived signatures must be indistinguishable from fresh signatures. For this reason we explicitly split updating and signing signatures: updating a signature does *not* draw a new tag.

Security Model. Next, we introduce the extended security model and define the notions of transparency, privacy, unforgeability, merge privacy, merge transparency, update privacy, and update transparency. We then show how these properties are related to each other. As before, we use the definitions given in [14,37,42,43] as our starting point.

As common in $\mathcal{RSS}$s, all of the following definitions specifically address the additional knowledge a third party can gain from the signature σ alone: if in real documents the redactions or updates are obvious due to additional context information or from the message contents itself, e.g., missing parts of a well known document structure, it may be trivial for attackers to detect them. This observation is general and also applies to schemes which offer context-hiding and cannot be avoided.

Unforgeability. No one must be able to produce a valid signature on a set $\mathcal{S}^*$ verifying under *pk* with elements not endorsed by the holder of *sk*, i.e., the signer. That is, even if an attacker can adaptively request signatures on different documents, and also can *adaptively update* them, it remains impossible to forge a signature for a new set or new elements not queried. In Fig. 1 we use $\mathcal{S}_{\tau^*}$ to remember all elements signed by the oracle under tag τ^* and $\mathcal{T}$ to collect all tags. This unforgeability definition is analogous to the standard unforgeability requirement of standard digital signature schemes [26]. We say that an $\mathcal{RSS}$ 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.

Privacy. The verifier should not be able to gain any knowledge about redacted elements without having access to them. In this definition, the adversary chooses two tuples $(\mathcal{S}_0, \mathcal{R}_0)$ and $(\mathcal{S}_1, \mathcal{R}_1)$, where $\mathcal{R}_i \subseteq \mathcal{S}_i$ describes what shall be removed from $\mathcal{S}_i$. 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

```
Experiment $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathcal{RSS}}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  Set $\mathrm{T} \leftarrow \emptyset$
  $(\mathcal{S}^*, \sigma^*, \tau^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda, sk, \cdot)}_{\mathsf{Update}(1^\lambda, sk, \cdot, \cdot, \cdot, \cdot)}(1^\lambda, pk)$
    For each query to oracle $\mathsf{Sign}$:
      let $(\mathcal{S}, \sigma, \tau)$ denote the answer from $\mathsf{Sign}$
      Set $\mathcal{S}_\tau \leftarrow \mathcal{S}$
      Set $\mathrm{T} \leftarrow \mathrm{T} \cup \{\tau\}$
    For each call to oracle $\mathsf{Update}$:
      let $(\mathcal{S}, \sigma, \tau)$ denote the answer from $\mathsf{Update}$
      Set $\mathcal{S}_\tau \leftarrow \mathcal{S}_\tau \cup \mathcal{S}$
  return 1, if
    $\mathsf{Verify}(1^\lambda, pk, \mathcal{S}^*, \sigma^*, \tau^*) = 1$ and
    $\tau^* \notin \mathrm{T}$ or $\mathcal{S}^* \nsubseteq \mathcal{S}_{\tau^*}$
```

Fig. 1. Unforgeability

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

Fig. 2. Privacy

```
Experiment $\mathsf{Transparency}_{\mathcal{A}}^{\mathcal{RSS}}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $b \xleftarrow{\$} \{0, 1\}$
  $d \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda, sk, \cdot), \mathsf{Sign/Redact}(1^\lambda, \cdot, \cdot, sk, b), \mathsf{Update}(1^\lambda, sk, \cdot, \cdot, \cdot, \cdot)}(1^\lambda, pk)$
    where oracle $\mathsf{Sign/Redact}$ for input $\mathcal{S}, \mathcal{R}$:
      if $\mathcal{R} \nsubseteq \mathcal{S}$, return $\perp$
      $(\mathcal{S}, \sigma, \tau) \leftarrow \mathsf{Sign}(1^\lambda, sk, \mathcal{S})$,
      $(\mathcal{S}', \sigma', \tau) \leftarrow \mathsf{Redact}(1^\lambda, pk, \mathcal{S}, \sigma, \mathcal{R}, \tau)$
      if $b = 1$:
        $(\mathcal{S}', \sigma', \tau) \leftarrow \mathsf{Sign}(1^\lambda, sk, \mathcal{S}')$
      return $(\mathcal{S}', \sigma', \tau)$
  return 1, if $b = d$
```

Fig. 3. Transparency

signs $\mathcal{S}_b$ and then redacts $\mathcal{R}_b$. The adversary wins, if it can decide which pair was used by the oracle as the input to create its corresponding output. This is similar to the standard indistinguishability notion for encryption schemes [25]. We say that an $\mathcal{RSS}$ is *private*, if for every PPT adversary $\mathcal{A}$ the probability that the game depicted in Fig. 2 returns 1, is negligibly close to $\frac{1}{2}$. Note, this definition does not capture unlinkability.

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 $(\mathcal{S}, \mathcal{R})$, where $\mathcal{R} \subseteq \mathcal{S}$ describes what shall be removed from $\mathcal{S}$. The pair is input for a "Sign/Redact" oracle that either signs and redacts elements (using Redact) or remove elements as a redaction would do $(\mathcal{S} \setminus \mathcal{R})$ before signing it. The adversary wins, if it can decide which way was taken. We say that an $\mathcal{RSS}$ is *transparent*, if for every PPT adversary $\mathcal{A}$, the probability that the game depicted in Fig. 3 returns 1, is negligibly close to $\frac{1}{2}$.

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

Fig. 4. Merge Privacy

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

Fig. 5. Merge Transparency

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 this definition, the adversary can choose three sets $\mathcal{S}, \mathcal{R}_0, \mathcal{R}_1$. The oracle $\mathsf{LoRMerge}$ signs $\mathcal{S}$ and then generates two signed redacted versions $\mathcal{S}' = \mathcal{S} \setminus \mathcal{R}_b$ and $\mathcal{S}'' = \mathcal{R}_b$. Then, it merges the signatures again. The adversary wins, if it can decide if $\mathcal{R}_0$ or $\mathcal{R}_1$ was first redacted from $\mathcal{S}$ and then merged back. We say that an $\mathcal{RSS}$ is *merge private*, if for every PPT adversary $\mathcal{A}$, the probability that the game depicted in Fig. 4 returns 1, is negligibly close to $\frac{1}{2}$.

Merge Transparency. If a set is given to a third party, the party should not be able to decide whether the set has been created only by Sign or through Sign and Merge. The adversary can choose one tuple $(\mathcal{S}, \mathcal{R})$ with $\mathcal{R} \subseteq \mathcal{S}$. This pair is input to a $\mathsf{Sign/Merge}$ oracle that signs the set $\mathcal{S}$ and either returns this set/signature pair directly ($b = 1$) or redacts the $\mathcal{S}$ into two signed "halves" $\mathcal{R}$ and $\mathcal{T}$ only to merge them together again and return the set/signature pair derived using Merge ($b = 0$). The adversary wins, if it can decide which way was taken. We say that an $\mathcal{RSS}$ is *merge transparent*, if for every PPT adversary $\mathcal{A}$, the probability that the game depicted in Fig. 5 returns 1, is negligibly close to $\frac{1}{2}$.

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.

Update Privacy. If an updated set is given to another third party, the party should not be able to derive which elements have been added. In the game, the adversary wins, if it can decide which elements were added after signature generation. In this definition, the adversary can choose three sets $\mathcal{S}, \mathcal{R}_0, \mathcal{R}_1$. The oracle $\mathsf{LoRUpdate}$ signs $\mathcal{S} \cup \mathcal{R}_b$ and then adds $\mathcal{R}_{b-1}$ to the signature. The adversary wins, if it can decide which set was used for the update. A scheme $\mathcal{RSS}$ is *update private*, if for every PPT adversary $\mathcal{A}$, the probability that the game depicted in Fig. 6 returns 1, is negligibly close to $\frac{1}{2}$.

```
**Experiment** Update Privacy$_{\mathcal{A}}^{\mathcal{RSS}}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $b \overset{\$}{\leftarrow} \{0,1\}$
  $d \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda, sk, \cdot), \mathsf{LoRUpdate}(1^\lambda, \cdot, \cdot, \cdot, sk, b)}_{\mathsf{Update}(1^\lambda, sk, \cdot, \cdot, \cdot, \cdot)}(1^\lambda, pk)$
    where oracle LoRUpdate for input $\mathcal{S}, \mathcal{R}_0, \mathcal{R}_1$:
      $(\mathcal{S}', \sigma'_{\mathcal{S}}, \tau) \leftarrow \mathsf{Sign}(1^\lambda, sk, \mathcal{S} \cup \mathcal{R}_b)$
      return $\mathsf{Update}(1^\lambda, sk, \mathcal{S}', \sigma'_{\mathcal{S}}, \mathcal{R}_{1-b}, \tau)$
  return 1, if $b = d$
```

Fig. 6. Update Privacy

```
**Experiment** Update Transparency$_{\mathcal{A}}^{\mathcal{RSS}}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $b \overset{\$}{\leftarrow} \{0,1\}$
  $d \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda, sk, \cdot), \mathsf{Sign/Update}(1^\lambda, \cdot, \cdot, sk, b)}_{\mathsf{Update}(1^\lambda, sk, \cdot, \cdot, \cdot, \cdot)}(1^\lambda, pk)$
    where oracle Sign/Update for input $\mathcal{S}, \mathcal{R}$:
      if $b = 1$:  $(\mathcal{S}', \sigma', \tau) \leftarrow \mathsf{Sign}(1^\lambda, sk, \mathcal{S} \cup \mathcal{R})$,
      if $b = 0$:  $(\mathcal{T}', \sigma'_{\mathcal{T}}, \tau) \leftarrow \mathsf{Sign}(1^\lambda, sk, \mathcal{S})$
                   $(\mathcal{S}', \sigma', \tau) \leftarrow \mathsf{Update}(1^\lambda, sk, \mathcal{T}', \sigma'_{\mathcal{T}}, \mathcal{R}, \tau)$
      return $(\mathcal{S}', \sigma', \tau)$
  return 1, if $b = d$
```

Fig. 7. Update Transparency

Update Transparency. A verifying party should not be able to decide whether the received set has been created by Sign or through Update. The adversary can choose one pair $(\mathcal{S}, \mathcal{R})$. This pair is input to a Sign/Update oracle that either signs the set $\mathcal{S} \cup \mathcal{R}$ $(b = 1)$ or signs $\mathcal{S}$ and then adds $\mathcal{R}$ using Update $(b = 0)$. The adversary wins, if it can decide which way was taken. We say that a scheme $\mathcal{RSS}$ is *update transparent*, 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 before, the notions of update transparency and update privacy are, on purpose, kept very similar to the notions of privacy and transparency due to their similar goals.

Definition 2 (Secure $\mathcal{RSS}$). *We call an $\mathcal{RSS}$ secure, if it is unforgeable, transparent, private, merge transparent, merge private, update private, and update transparent.*

We now give some relations between the security properties. This section can be kept brief, as we tailored the definitions to be similar (in terms of relation) to the ones given in [14]. This is intentional, to keep consistent with existing wording and to blend into the large body of existing work. We have to explicitly consider the update-oracle, as it may leak information about the secret key sk.

Theorem 1 (Merge Transparency $\implies$ Merge Privacy). *Every scheme which is merge transparent, is also merge private.*

Proof. Intuitively, the proof formalizes the following idea: if an adversary can decide which elements have been merged, then it can decide that the signature cannot be created by Sign, but by Merge.

Assume an (efficient) adversary $\mathcal{A}$ that wins our merge privacy with probability $\frac{1}{2} + \epsilon$. We can then construct an (efficient) adversary $\mathcal{B}$ which wins the merge transparency game with probability $\frac{1}{2} + \frac{\epsilon}{2}$. According to the merge transparency game, $\mathcal{B}$ receives a public key pk and oracle access to $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{O}^{\mathsf{Sign/Merge}}$, and $\mathcal{O}^{\mathsf{Update}}$. Let $\mathcal{B}$ randomly pick a bit $b' \in \{0,1\}$. $\mathcal{B}$ forwards pk to $\mathcal{A}$. Whenever $\mathcal{A}$ requests access to the signing oracle $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{B}$ honestly forwards the query to its oracle and returns the unmodified answer to $\mathcal{A}$. The same is true for $\mathcal{O}^{\mathsf{Update}}$. When $\mathcal{A}$ requests access to $\mathcal{O}^{\mathsf{LoRMerge}}$, i.e., when it sends a query $(\mathcal{S}, \mathcal{R}_0, \mathcal{R}_1)$, then $\mathcal{B}$ checks that $\mathcal{R}_0 \subset \mathcal{S} \wedge \mathcal{R}_1 \subset \mathcal{S}$ and forwards $(\mathcal{S}, \mathcal{R}_{b'})$ to $\mathcal{O}^{\mathsf{Sign/Merge}}$ and returns the answer to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs its guess d. Our adversary $\mathcal{B}$ outputs 0, if $d = b'$ and 1 otherwise. What is the probability that $\mathcal{B}$ is correct? We have to consider two cases:

1. If $b = 0$, then $\mathcal{O}^{\mathsf{Sign/Merge}}$ signs, redacts, and merges the set. This gives exactly the same answer as $\mathcal{O}^{\mathsf{LoRRedact}}$ would do, if using the bit b'. Hence, $\mathcal{A}$ can correctly guess the bit b' with probability at least $\frac{1}{2} + \epsilon$, if $b = 0$.
2. If $b = 1$, then $\mathcal{O}^{\mathsf{Sign/Merge}}$ always signs the set as is. Hence, the answer is independent of b'. $\Pr[\mathcal{B} = 1 \mid b = 1] = \frac{1}{2}$ follows.

Hence, due to the probability of $\frac{1}{2}$ that $b = 1$, it follows that $\Pr[\mathcal{B} = b] = \frac{1}{2} + \frac{\epsilon}{2}$. Hence, $\mathcal{B}$ has non-negligible advantage, if ϵ is non-negligible.

Theorem 2 (Merge Privacy $\not\Rightarrow$ Merge Transparency). *There is a scheme which is merge private, but not merge transparent.*

Proof. At sign, we append a bit $d = 0$. For all other algorithms d is cut off, and appended after the algorithm finished. However, we set $d = 1$ once signatures are merged. Obviously, we leave all other properties intact.

Theorem 3 (Update Transparency $\Longrightarrow$ Update Privacy). *Every scheme which is update transparent, is also update private.*

Proof. The proof is essentially the same as for Th. 1.

Theorem 4 (Update Privacy $\not\Rightarrow$ Update Transparency). *There is a scheme which is update private, but not update transparent.*

Proof. The proof is essentially the same as for Th. 2.

Theorem 5 (Merge Transparency is independent). *There is a scheme which fulfills all mentioned security goals but merge transparency.*

Proof. The proof is essentially the same as for Th. 2.

Theorem 6 (Update Transparency is independent). *There is a scheme which fulfills all mentioned security goals but update transparency.*

Proof. The proof is essentially the same as for Th. 2.

Theorem 7 (Unforgeability is independent). *There is a scheme which fulfills all mentioned security goals but unforgeability.*

Proof. We simply use a verify algorithm which always accepts all inputs.

Theorem 8 (Transparency $\Longrightarrow$ Privacy). *Every scheme which is transparent, is also private. Similar to [14].*

Theorem 9 (Privacy $\not\Rightarrow$ Transparency). *There is a scheme which is private, but not transparent. Similar to [14].*

Theorem 10 (Transparency is independent). *There is a scheme which fulfills all mentioned security goals but transparency. Similar to [14].*

Even though the transparency properties give stronger security guarantees, legislation requires that altered signatures must be distinguishable from new ones [16]. However, privacy is the absolute minimum to be useful [16]. We therefore need to split the definitions: depending on the use-case, one can then decide which properties are required.

3 Trapdoor-Accumulators and Constructions

Cryptographic accumulators have been introduced by *Benaloh* and *de Mare* [9]. They hash a potentially very large set $\mathcal{S}$ into a short single value a, called the accumulator. For each element accumulated, a witness is generated, which vouches for the accumulation. A trapdoor-accumulator allows generating proofs for new elements not contained by use of a trapdoor. Our construction is based upon such an accumulator. Using an accumulator allows us to achieve mergeability "for free", as we can add and remove witnesses and the corresponding elements freely. We do not require non-membership witnesses [32], or non-deniability [34] for our scheme to work. We do note that there exists the possibility of dynamically updating an accumulator [18]. However, they also allow removing accumulated elements, while they need to adjust every single witness. This is not necessary for our goals. However, accumulators are very versatile. We leave it as open work to discuss the impact of accumulators with different properties plugged into our construction.

Algorithmic Description and Security Model. We now introduce trapdoor accumulators. The definition is derived from [6].

Definition 3 (Trapdoor Cryptographic Accumulators). *A cryptographic trapdoor accumulator $\mathcal{ACC}$ consists of four efficient (PPT) algorithms. In particular, $\mathcal{ACC} := (\mathsf{Gen}, \mathsf{Dig}, \mathsf{Proof}, \mathsf{Verf})$ such that:*

Gen. *The algorithm* Gen *is the key generator. On input of the security parameter λ, it outputs the key pair $(\mathrm{sk}_{\mathcal{ACC}}, \mathrm{pk}_{\mathcal{ACC}}) \leftarrow \mathsf{Gen}(1^\lambda)$*

Dig. *The algorithm Dig takes as input the set $\mathcal{S}$ to accumulate, the public parameters $\mathrm{pk}_{\mathcal{ACC}}$. It outputs an accumulator value $a \leftarrow \mathsf{Dig}(1^\lambda, \mathrm{pk}_{\mathcal{ACC}}, \mathcal{S})$*

Proof. *The deterministic algorithm* Proof *takes as input the secret key $\mathrm{sk}_{\mathcal{ACC}}$, the accumulator a, and a value v and returns a witness p for v. Hence, it outputs $p \leftarrow \mathsf{Proof}(1^\lambda, \mathrm{sk}_{\mathcal{ACC}}, a, v)$*

Verf. *The verification algorithm* Verf *takes as input the public key $\mathrm{pk}_{\mathcal{ACC}}$, an accumulator a, a witness p, and a value v and outputs a bit $d \in \{0, 1\}$, indicating whether p is a valid witness for v w.r.t. a and $\mathrm{pk}_{\mathcal{ACC}}$. Hence, it outputs $d \leftarrow \mathsf{Verf}(1^\lambda, \mathrm{pk}_{\mathcal{ACC}}, a, v, p)$*

We require the usual correctness properties to hold. Refer to [6] for a formal definition of the correctness properties for accumulators.

```
Experiment $\mathsf{Strong-Coll.-Res.}^{\mathcal{ACC}}_{\mathcal{A}}(\lambda)$
  $(sk_{\mathcal{ACC}}, pk_{\mathcal{ACC}}) \leftarrow \mathsf{Gen}(1^\lambda)$
  $(S^*, st) \leftarrow \mathcal{A}_1(1^\lambda, pk_{\mathcal{ACC}})$ //$st$ denotes $\mathcal{A}$'s state
  $a \leftarrow \mathsf{Dig}(1^\lambda, pk_{\mathcal{ACC}}, S^*)$
  $(v^*, p^*) \leftarrow \mathcal{A}_2^{\mathsf{Proof}(1^\lambda, sk_{\mathcal{ACC}}, a, \cdot)}(st, a)$
  return 1, if
    $\mathsf{Verf}(1^\lambda, pk_{\mathcal{ACC}}, a, v^*, p^*) = 1$,
    and $v^*$ has not been queried to $\mathsf{Proof}$
```

Fig. 8. Strong Collision-Resistance

Strong Collision-Resistance. An adversary should not be able find a valid witness/element pair (p^*, v^*) for a given accumulator a, even if it is allowed to adaptively query for elements not contained in the original set accumulated and to choose the original set to be accumulated. We call a family of trapdoor accumulators *strongly collision-resistant*, if the probability that the experiment depicted in Fig. 8 returns 1, is negligible. We do note that this definition is very similar to the standard unforgeability of signature schemes. The naming is due to historical reasons [6].

Trapdoor-Accumulators. Next, we show how a trapdoor-accumulator can be build. We use the ideas given in [6], but make use of the trapdoor $\varphi(n)$.

Construction 1 (Trapdoor-Accumulator $\mathcal{ACC}$). *We require a division-intractable hash-function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ mapping to odd numbers. A formal definition is given in [24]. Let $\mathcal{ACC} := (\mathsf{Gen}, \mathsf{Dig}, \mathsf{Proof}, \mathsf{Verf})$ such that:*

Gen. *Generate $n = pq$, where p and q are distinct safe primes of length λ.*[3] *Return $(\varphi(n), (n, \mathcal{H}))$, where $\varphi(pq) := (p-1) \cdot (q-1)$.*

Dig. *To improve efficiency, we use the build-in trapdoor. A new digest can therefore be drawn at random. Return $a \in_R \mathbb{Z}_n^\times$.*

Proof. *To generate a witness p_i for an element v_i, set $v_i' \leftarrow \mathcal{H}(v_i)$. Output $p_i \leftarrow a^{v_i'^{-1} \pmod{\varphi(n)}} \bmod n$*

Verf. *To check the correctness of a proof p w.r.t. an accumulator a, the public key $\mathrm{pk}_{\mathcal{ACC}}$, and a value v, output 1, if $a \overset{?}{=} p^{\mathcal{H}(v)} \pmod{n}$, and 0 otherwise*

We do note that this construction is related to GHR-signatures [24]. Due to the build-in trapdoor, we do not require any auxiliary information as proposed in [6]. The use of safe primes allows us to almost always find a root for odd numbers. If we are not able to do so, we can trivially factor n. The proofs that our trapdoor-accumulator is strongly collision-resistant can be found in the appendix.

We want to explicitly stress that an adversary can simulate the Proof-oracle itself for the elements used for Dig. It calculates $a = x^{\prod_{v_i \in S} \mathcal{H}(v_i)} \bmod n$ for a random $x \in_R \mathbb{Z}_n^\times$ and for each proof p_i, it lets $p_i = x^{\prod_{v_j \in S, i \neq j} \mathcal{H}(v_j)} \bmod n$. For new elements, this technique does not work. Note, a is drawn at random

[3] A prime p is safe, if $p = 2p' + 1$, where p' is also prime.

for efficiency. We can also use the slower method aforementioned: a will be distributed exactly in the same way.

Updatable and Mergeable $\mathcal{RSS}$ — Construction. The basic ideas are: (1) Our trick is to fix the accumulator a for *all* signatures. Additionally, each element is tagged with a unique string τ to tackle mix-and-match attacks. Hence, all derived subset/signature pairs are linkable by the tag τ. τ is also accumulated to avoid trivial "empty-set"-attacks. (2) Redactions remove v_i and its corresponding witness p_i. The redactions are private, as without knowledge of the proof p_i nobody can verify if v_i is "in" the accumulator a. (3) Mergeability is achieved, as supplying an element/witness pair allows a third party to add it back into the signature. (4) Unforgeability comes from the strong collision-resistance of $\mathcal{ACC}$. (5) Dynamic updates are possible due to a trapdoor in $\mathcal{ACC}$, only known to the signer. (6) Privacy directly follows from definitions, i.e., the number of proofs is fixed, while the proofs itself are deterministically generated, without taking already generated proofs into account. We do note that we can also use aggregate-signatures to reduce the signature size [11]. However, we want to show that an accumulator is enough to build $\mathcal{RSS}$s. Having a suitable security model, we can now derive an efficient, stateless, yet simple construction. Our construction is inspired by [28]. However, their construction is forgeable and non-private in our model, as they allow for arbitrary merging, and do not hide redacted elements completely. One may argue that a very straight-forward construction exists: one signs each element $v_i \in \mathcal{S}$ and gives out the signatures. However, our approach has some advantages: we can exchange the accumulator to derive new properties, e.g., prohibiting updates using a trapdoor-free accumulator [34]. Moreover, we prove that using accumulators are sufficient, opposing the results of [30].

Construction 2 (Updatable and Mergeable $\mathcal{RSS}$). *We use $||$ to denote a uniquely reversible concatenation of strings. Let $\mathcal{RSS}$:= (KeyGen, Sign, Verify, Redact, Update, Merge) such that:*

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

1. *Generate key pair required for $\mathcal{ACC}$, i.e., run $(\mathrm{sk}_{\mathcal{ACC}}, \mathrm{pk}_{\mathcal{ACC}}) \leftarrow \mathsf{Gen}(1^\lambda)$*
2. *Call $a \leftarrow \mathsf{Dig}(\mathrm{pk}_{\mathcal{ACC}}, \emptyset)$*
3. *Output $(\mathrm{sk}_{\mathcal{ACC}}, (\mathrm{pk}_{\mathcal{ACC}}, a))$*

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

1. *Draw a tag $\tau \in_R \{0,1\}^\lambda$*
2. *Let $p_\tau \leftarrow \mathsf{Proof}(\mathrm{sk}_{\mathcal{ACC}}, a, \tau)$*
3. *Output $(\mathcal{S}, \sigma, \tau)$, where $\sigma = (p_\tau, \{(v_i, p_i) \mid v_i \in \mathcal{S} \wedge p_i \leftarrow \mathsf{Proof}(\mathrm{sk}_{\mathcal{ACC}}, a, v_i||\tau)\})$*

Verify. *To verify signature $\sigma = (p_\tau, \{(v_1, p_1), \ldots, (v_k, p_k)\})$ with tag τ, perform:*

1. *For all $v_i \in \mathcal{S}$ check that $\mathsf{Verf}(\mathrm{pk}_{\mathcal{ACC}}, a, v_i||\tau, p_i) = 1$*
2. *Check that $\mathsf{Verf}(\mathrm{pk}_{\mathcal{ACC}}, a, \tau, p_\tau) = 1$*
3. *If Verf succeeded for all elements, output 1, otherwise 0*

Redact. *To redact a subset $\mathcal{R}$ from a valid signed set $(\mathcal{S}, \sigma)$ with tag τ, with $\mathcal{R} \subseteq \mathcal{S}$, the algorithm performs the following steps:*

1. *Check the validity of σ using* Verify*. If σ is not valid, return $\perp$*
2. *Output $(\mathcal{S}', \sigma', \tau)$, where $\sigma' = (p_\tau, \{(v_i, p_i) \mid v_i \in \mathcal{S} \setminus \mathcal{R}\})$*

Update. *To update a valid signed set $(\mathcal{S}, \sigma)$ with tag τ by adding $\mathcal{U}$ and knowing $\mathrm{sk}_{\mathcal{ACC}}$, the algorithm performs the following steps:*

1. *Verify σ w.r.t. τ using* Verify*. If σ is not valid, return $\perp$*
2. *Output $(\mathcal{S} \cup \mathcal{U}, \sigma', \tau)$, where $\sigma' = (p_\tau, \{(v_i, p_i) \mid v_i \in \mathcal{S}\} \cup \{(v_k, p_k) \mid v_k \in \mathcal{U}, p_k \leftarrow \mathsf{Proof}(\mathrm{sk}_{\mathcal{ACC}}, a, v_k \| \tau)\})$*

Merge. *To merge two valid set/signature pairs $(\mathcal{S}, \sigma_\mathcal{S})$ and $(\mathcal{T}, \sigma_\mathcal{T})$ with an equal tag τ, the algorithm performs the following steps:*

1. *Verify $\sigma_\mathcal{S}$ and $\sigma_\mathcal{T}$ w.r.t. τ using* Verify*. If they do not verify, return $\perp$*
2. *Check, that both have the same tag τ*
3. *Output $(\mathcal{S} \cup \mathcal{T}, \sigma_\mathcal{U}, \tau)$, where $\sigma_\mathcal{U} = (p_\tau, \{(v_i, p_i) \mid v_i \in \mathcal{S} \cup \mathcal{T}\})$, where p_i is taken from the corresponding signature*

Our construction is elegantly simple, yet fulfills all security goals (all but unforgeability even perfectly), and is therefore useable in practice. The proofs of security are in the appendix. All reductions are tight, i.e., we have no reduction losses. We want to explicitly clarify that we do not see the transitive closure of the updates as forgeries. If we want to disallow the "transitive update merging", we can deploy accumulators which also update the witnesses, e.g., [18]. This requires a new security model, which renders existing constructions insecure, which we wanted to avoid. We leave this as future work.

4 Conclusion and Open Questions

We have revised existing notions of redactable signature schemes. We derived a security model, addressing the shortcomings of existing ones. We presented an attack on existing $\mathcal{RSS}$s, if dynamic updates are not carefully considered. Moreover, we have formalized the notion of mergeability, the inverse of redactions. These properties allow using $\mathcal{RSS}$s in more application scenarios, e.g., distributed databases and general cloud-storage. Finally, we have presented a provably secure construction in the standard model, based on a novel trapdoor-accumulator. It is unclear how we can prohibit dynamic updates and merging signatures, how accumulators and signatures are related to each other, and if efficient constructions for more complex data-structures exist.

References

1. Ahn, J.H., Boneh, D., Camenisch, J., Hohenberger, S., Shelat, A., Waters, B.: Computing on authenticated data. ePrint Report 2011/096 (2011)
2. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable signatures. In: de Capitani di Vimercati, S., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 159–177. Springer, Heidelberg (2005)
3. Attrapadung, N., Libert, B., Peters, T.: Computing on authenticated data: New privacy definitions and constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 367–385. Springer, Heidelberg (2012)

4. Attrapadung, N., Libert, B., Peters, T.: Efficient completely context-hiding quotable and linearly homomorphic signatures. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 386–404. Springer, Heidelberg (2013)
5. Backes, M., Meiser, S., Schröder, D.: Delegatable functional signatures. IACR Cryptology ePrint Archive, 408 (2013)
6. Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 480–494. Springer, Heidelberg (1997)
7. Becker, A., Jensen, M.: Secure combination of xml signature application with message aggregation in multicast settings. In: ICWS, pp. 531–538 (2013)
8. Bellare, M., Goldreich, O., Goldwasser, S.: Incremental cryptography: The case of hashing and signing. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 216–233. Springer, Heidelberg (1994)
9. Benaloh, J.C., de Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 274–285. Springer, Heidelberg (1994)
10. Boneh, D., Freeman, D.M.: Homomorphic signatures for polynomial functions. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 149–168. Springer, Heidelberg (2011)
11. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)
12. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. IACR Cryptology ePrint Archive 401 (2013)
13. Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of Sanitizable Signatures Revisited. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 317–336. Springer, Heidelberg (2009)
14. Brzuska, C., et al.: Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 87–104. Springer, Heidelberg (2010)
15. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Unlinkability of sanitizable signatures. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 444–461. Springer, Heidelberg (2010)
16. Brzuska, C., Pöhls, H.C., Samelin, K.: Non-Interactive Public Accountability for Sanitizable Signatures. In: De Capitani di Vimercati, S., Mitchell, C. (eds.) EuroPKI 2012. LNCS, vol. 7868, pp. 178–193. Springer, Heidelberg (2013)
17. Brzuska, C., Pöhls, H.C., Samelin, K.: Efficient and Perfectly Unlinkable Sanitizable Signatures without Group Signatures. In: Katsikas, S., Agudo, I. (eds.) EuroPKI 2013. LNCS, vol. 8341, pp. 12–30. Springer, Heidelberg (2014)
18. Camenisch, J.L., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002)
19. Chang, E.-C., Lim, C.L., Xu, J.: Short Redactable Signatures Using Random Trees. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 133–147. Springer, Heidelberg (2009)
20. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable signatures: Complex unary transformations and delegatable anonymous credentials. IACR Cryptology ePrint Archive 179 (2013)
21. de Meer, H., Pöhls, H.C., Posegga, J., Samelin, K.: Scope of security properties of sanitizable signatures revisited. In: ARES, pp. 188–197 (2013)

22. de Meer, H., Pöhls, H.C., Posegga, J., Samelin, K.: On the relation between redactable and sanitizable signature schemes. In: Jürjens, J., Piessens, F., Bielova, N. (eds.) ESSoS. LNCS, vol. 8364, pp. 113–130. Springer, Heidelberg (2014)
23. Deiseroth, B., Fehr, V., Fischlin, M., Maasz, M., Reimers, N.F., Stein, R.: Computing on authenticated data for adjustable predicates. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 53–68. Springer, Heidelberg (2013)
24. Gennaro, R., Halevi, S., Rabin, T.: Secure hash-and-sign signatures without the random oracle. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 123–139. Springer, Heidelberg (1999)
25. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. 28(2), 270–299 (1984)
26. Goldwasser, S., Micali, S., Rivest, R.L.: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. SIAM JoC 17, 281–308 (1988)
27. Herkenhöner, R., Jensen, M., Pöhls, H.C., De Meer, H.: Towards automated processing of the right of access in inter-organizational web service compositions. In: IEEE WSBPS 2010. IEEE (July 2010)
28. Johnson, R., Molnar, D., Song, D., Wagner, D.: Homomorphic signature schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 244–262. Springer, Heidelberg (2002)
29. Kiltz, E., Mityagin, A., Panjwani, S., Raghavan, B.: Append-only signatures. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 434–445. Springer, Heidelberg (2005)
30. Kundu, A., Bertino, E.: Privacy-preserving authentication of trees and graphs. Int. J. Inf. Sec. 12(6), 467–494 (2013)
31. Lai, J., Ding, X., Wu, Y.: Accountable trapdoor sanitizable signatures. In: Deng, R.H., Feng, T. (eds.) ISPEC 2013. LNCS, vol. 7863, pp. 117–131. Springer, Heidelberg (2013)
32. Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 253–269. Springer, Heidelberg (2007)
33. Lim, S., Lee, E., Park, C.-M.: A short redactable signature scheme using pairing. SCN 5(5), 523–534 (2012)
34. Lipmaa, H.: Secure accumulators from euclidean rings without trusted setup. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 224–240. Springer, Heidelberg (2012)
35. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)
36. Miyazaki, et al.: Digital documents sanitizing problem. Institute of Electronics, Information and Communication Engineers Technical Reports 103(195), 61–67 (2003)
37. Miyazaki, K., Hanaoka, G., Imai, H.: Digitally signed document sanitizing scheme based on bilinear maps. In: ASIACCS 2006, pp. 343–354. ACM (2006)
38. Pöhls, H.C.: Verifiable and revocable expression of consent to processing of aggregated personal data. In: Chen, L., Ryan, M.D., Wang, G. (eds.) ICICS 2008. LNCS, vol. 5308, pp. 279–293. Springer, Heidelberg (2008)
39. Pöhls, H.C., Bilzhause, A., Samelin, K., Posegga, J.: Sanitizable signed privacy preferences for social networks. In: DICCDI 2011. LNI. GI (2011)

40. Pöhls, H.C., Samelin, K., Posegga, J.: Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 166–182. Springer, Heidelberg (2011)
41. Rass, S., Slamanig, D.: Cryptography for Security and Privacy in Cloud Computing. Artech House (2013)
42. Samelin, K., Pöhls, H.C., Bilzhause, A., Posegga, J., de Meer, H.: On Structural Signatures for Tree Data Structures. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 171–187. Springer, Heidelberg (2012)
43. Samelin, K., Pöhls, H.C., Bilzhause, A., Posegga, J., de Meer, H.: Redactable signatures for independent removal of structure and content. In: Ryan, M.D., Smyth, B., Wang, G. (eds.) ISPEC 2012. LNCS, vol. 7232, pp. 17–33. Springer, Heidelberg (2012)
44. Slamanig, D., Rass, S.: Generalizations and extensions of redactable signatures with applications to electronic healthcare. In: De Decker, B., Schaumüller-Bichl, I. (eds.) CMS 2010. LNCS, vol. 6109, pp. 201–213. Springer, Heidelberg (2010)
45. Steinfeld, R., Bull, L., Zheng, Y.: Content extraction signatures. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 285–304. Springer, Heidelberg (2002)
46. Wu, Z.-Y., Hsueh, C.-W., Tsai, C.-Y., Lai, F., Lee, H.-C., Chung, Y.: Redactable Signatures for Signed CDA Documents. Journal of Medical Systems, 1–14 (December 2010)

A Security Proofs

Theorem 11 (Our Construction is Unforgeable). *Our construction is unforgeable, if the underlying accumulator is strongly collision-resistant.*

Proof. We do not consider tag collisions, as they only appear with negligible probability. $S^* \subseteq S_\tau$ for some signed τ is not a forgery, but a redaction. We denote the adversary winning the unforgeability game as $\mathcal{A}$. We can now derive that the forgery must fall into exactly one of the following categories:

Case 1: $\mathcal{S}^* \nsubseteq \mathcal{S}_{\tau^*}$, and τ^* was used as a tag by Sign
Case 2: $\mathcal{S}^*$ verifies, and τ^* was never used as a tag by Sign

Each case leads to a contradiction about the security of our accumulator.

Case 1. In this case, an element v^* not been returned by the Proof-oracle for the accumulator a, but is contained in $\mathcal{S}^*$. We break the strong collision-resistance of the underlying accumulator by letting $\mathcal{B}$ use $\mathcal{A}$ as a black-box:

1. $\mathcal{B}$ receives $pk_{\mathcal{ACC}}$ from the challenger
2. $\mathcal{B}$ requests an accumulator a for $\emptyset$
3. $\mathcal{B}$ receives a from its own challenger
4. $\mathcal{B}$ forwards $pk = (pk_{\mathcal{ACC}}, a)$ to $\mathcal{A}$
5. For each query to the signing oracle, $\mathcal{B}$ answers it honestly: it draws τ honestly and uses the Proof-oracle provided to get a witness for each $v_j \in \mathcal{S}_i$ queried, with τ concatenated as the label. Also, $\mathcal{B}$ gets a proof for τ

6. For each call to the Update-oracle, $\mathcal{B}$ uses its Proof-oracle provided to get a witness for each $v_j \in \mathcal{S}_i$ queried, with τ concatenated as the label
7. Eventually, $\mathcal{A}$ outputs a pair (S^*, σ^*)
8. $\mathcal{B}$ looks for (v^*, p^*), v^* not queried to Proof, in (S^*, σ^*) and returns them

In other words, there exists an element $v^* \in \mathcal{S}^*$ with a corresponding witness p^*. If v^* has not been asked to the Proof-oracle, $\mathcal{B}$ breaks the collision-resistance of the underlying accumulator by outputting (v^*, p^*). This happens with the same probability as $\mathcal{A}$ breaks unforgeability in case 1. Hence, the reduction is tight.

Case 2. In case 2, the tag τ^* has not been accumulated. We break the strong collision-resistance of the underlying accumulator by letting $\mathcal{B}$ use $\mathcal{A}$:

1. $\mathcal{B}$ receives $pk_{\mathcal{ACC}}$ from the challenger
2. $\mathcal{B}$ requests an accumulator a for $\emptyset$
3. $\mathcal{B}$ forwards $pk = (pk_{\mathcal{ACC}}, a)$ to $\mathcal{A}$
4. For each query to the signing oracle, $\mathcal{B}$ answers it honestly: it draws τ honestly and uses the Proof-oracle provided to get a witness for each $v_j \in \mathcal{S}_i$ queried, with τ concatenated as the label. Also, $\mathcal{B}$ gets a proof for τ
5. For calls to the Update-oracle, $\mathcal{B}$ uses its Proof-oracle provided to get a witness for each $v_j \in \mathcal{S}_i$ queried, with τ concatenated as the label
6. Eventually, $\mathcal{A}$ outputs a pair (S^*, σ^*, τ^*)
7. $\mathcal{B}$ returns (p^*_τ, τ^*). Both is contained in σ^*

In other words, there exists an element $\tau^* \in \sigma^*$ with a corresponding witness p^*_τ, as otherwise σ^* would not verify. We know that τ^* was not queried to Proof, because otherwise we have case 1. This happens with the same probability as $\mathcal{A}$ breaks the unforgeability in case 2. Note, we can ignore additional elements here. Again, the simulation is perfect.

Theorem 12 (Our Construction is Merge Private and Transparent). *Our construction is merge private and merge transparent.*

Proof. The distributions of merged and freshly signed signatures are *equal.* In other words, the distributions are the same. This implies, that our construction is *perfectly* merge private and *perfectly* merge transparent.

Theorem 13 (Our Construction is Transparent and Private).

Proof. As the number of proofs only depends on n, which are also deterministically generated, without taking existing proofs into account, an adversary has zero advantage on deciding how many additional proofs have been generated. Moreover, redacting only removes elements and proofs from the signatures. Hence, fresh and redacted signatures are distributed *identically. Perfect* transparency, and therefore also *perfect* privacy, is implied.

Theorem 14 (Our Construction is Update Private and Transparent). *Our construction is update private and update transparent.*

Proof. The distributions of updated and freshly signed signatures are *equal*. In other words, the distributions are the same. This implies, that our construction is *perfectly* update private and *perfectly* update transparent.

Theorem 15 (The Accumulator is Strongly Collision-Resistant).

Proof. Let $\mathcal{A}$ be an adversary breaking the strong-collision-resistance of our accumulator. We can then turn $\mathcal{A}$ into an adversary $\mathcal{B}$ which breaks the unforgeability of the GHR-signature [24] in the following way:

1. $\mathcal{B}$ receives the modulus n, the hash-function $\mathcal{H}$, and the value s. All is provided by the GHR-challenger
2. $\mathcal{B}$ sends $pk = (n, \mathcal{H})$ to $\mathcal{A}$. Then, $\mathcal{B}$ waits for $\mathcal{S}$ from $\mathcal{A}$
3. $\mathcal{B}$ sends s to $\mathcal{A}$. Note, we have a *perfect* simulation here, even as we ignore $\mathcal{S}$, as the GHR-signature scheme draws s in the *exact* same way as we do for our accumulator
4. For each Proof-oracle query v_i, $\mathcal{B}$ asks its signing oracle provided, which returns a signature σ_i. Send σ_i as the witness p_i back to $\mathcal{A}$
5. Eventually, $\mathcal{A}$ comes up with an attempted forgery (v^*, p^*)
6. $\mathcal{B}$ returns (v^*, p^*) as its own forgery attempt

Now let $y = v^*$, and $p = \sigma^*$. As $s = p^{\mathcal{H}(y)} \pmod{n}$, and we have embedded our challenges accordingly, $\mathcal{B}$ breaks the GHR-signature with the same probability as $\mathcal{A}$ breaks the strong collision-resistance of our trapdoor-accumulator. [24] shows how to break the strong-RSA-assumption with the given forgery.