Accountable Redactable Signatures

Henrich C. Pöhls* and Kai Samelin‡
*Chair of IT-Security and Institute of IT-Security and Security Law (ISL), University of Passau
‡IBM Research Zurich and Technical University of Darmstadt
Email: hp@sec.uni-passau.de, ksa@zurich.ibm.com

***Abstract*—Redactable signature schemes (RSS) allow removing blocks from signed data. State-of-the-art schemes have public redactions, i.e., any party can remove parts from a signed message. This prohibits meaningful definitions of accountability. We address this gap by introducing the notion of accountable redactable signature schemes (ARSS). We present a generic construction which couples a sanitizable signature scheme (SSS) to profit from its accountability with an RSS to maintain the reduced malleability of RSSs. Depending on the building blocks, the resulting scheme offers transparency or public accountability. Transparency provides stronger privacy guarantees, while public accountability meets legal and application requirements.**

I. Introduction

A redactable signature scheme (RSS) allows removing blocks $m[i]$ from a signed message $m = (m[1], \ldots, m[\ell])$. Hence, a redaction of the block $m[i]$ results in a redacted message $m' = (m[1], \ldots, m[i-1], m[i+1], \ldots, m[\ell])$. The derived signature σ' still verifies under the original signer's public key. This separates them from standard digital signatures, which prohibit any alteration of signed messages. 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 re-sign a message every time an alteration is necessary. Applications include privacy-preserving handling of patient data [5], [6], [35], [37], social networks [30], and "Blank Signatures" [19], [24]. As an example, identifying information inside patient records can be redacted: the remaining information is still enough for an accountant. This protects the patients' privacy. However, nobody is able to alter the given message in an arbitrary way, i.e., only *removal* of blocks leads to a valid signature. RSSs thus address the "digital message sanitization problem" [29]. Real implementations are given in [17], [32], [35], [37]. Contrary to sanitizable signatures (SSS) [2], [7], which allow to alter some blocks to arbitrary bitstrings (but no redaction), accountability of RSSs has not been fully discussed in the literature; to achieve legal recognition of RSSs this has to be addressed [11], [18]. In a nutshell, accountability allows to derive the accountable party of a signature/message pair (σ, m). This paper shows how to make RSSs accountable.

1) Example: Assume that patients' electronic health record contain prescribed treatments, e.g., the amount of infusions. For some treatments an apparatus, e.g., a smart pump, reads details from a record and carries out the treatment. Hence, health record data must be protected against malicious modifications, e.g., no wrong dosage. Assume further that the hospital offers standard vaccination, but also treats patients with severe diseases, e.g., multistage measles, avian influenza or even ebola. Thus, patients want to selectively show treatments to other institutions, e.g., to their employer only the recent treatment for vaccination, even if they have had other treatments. The doctor signs treatments along with patient's name and an internal identifier for him. The signed message is stored on a hospital's server and used by medical apparatus. Information about treatments is given to other medical staff, accountants, and each patient. Assume finally, that patients use signed hospital treatments to get reimbursed by the insurance.

RSSs allow to hand over only selected but integrity protected information. For example, a hospital server redacts patients' names before giving the records to the hospital's accountant [37], patients retrieve their own health record and redact it themselves before giving it to their employer. However, standard RSSs allow redactions by anyone, including the accountant, without detecting who redacted what. For example, the accountant teaming up with the patient redacts treatments to charge less to the patient's health insurance. This situation is clearly not acceptable. In fact, many electronic solutions for real-world applications explicitly deploy electronic signatures to generate legal evidence against the accountable party.

2) Motivation and Contribution: As aforementioned, accountability, i.e., it can be derived which party is accountable for a given signature/message pair (σ, m), has only been defined for SSSs [7] yet. We present a publicly accountable RSS and a transparent RSS. The former allows to reproducibly derive the accountable party without knowing any secrets and without any additional protocol interaction. Public accountability was shown by *Brzuska* et al. and *Höhne* et al. to be required to reach a level of legal evidence equal to standard digital signatures [11], [25]. This legal status is of paramount importance to actually deploy RSSs in real applications. Transparency, i.e., the anonymity of the accountable party, contradicts some requirements from the legal side [11]. However, transparency offers additional privacy guarantees which are required if the existence of a redaction must be hidden, e.g., if discrimination may follow [11]. We formalize both types of accountability to cover both use-cases. Our contribution is therefore manifold: (1) We present a formal framework for accountable RSSs. (2) We formally define the security definitions of accountable RSSs. (3) Our security notions are "compatible" with existing ones, i.e., they can be related to prior definitions. (4) We introduce a new generic construction. Depending on the properties of the underlying building blocks, it achieves public accountability or transparency. (5) We prove our construction tightly secure, i.e., the reduction loss is constant. (6) The construction, shown in Sect. IV, relies on standard building blocks and can easily be extended to multi-sanitizer environments [14]. (7) Our construction exclusively covers messages represented as lists of blocks. However, our proofs do not depend on this structure, thus we are confident that our ideas can be extended easily to more complex data-structures.

3) State-of-the-Art: On the one hand, RSSs have been introduced in [26] and, in a different notion, in [36]. Based on these first ideas, additional work appeared. Some cover more complex data-structures like trees [8], [33], and graphs [28]. The standard security properties of RSSs have been formalized in [8], [16], [31], [33]. Recently, *Ahn* et al. introduced the notion of statistically unlinkable RSSs [1]. Even stronger privacy notions have lately been discussed in [3], [4]. The latter schemes only allow for quoting instead of arbitrary redactions, i.e., redactions are only allowed at the beginning and at the end of a message. Moreover, the scheme by *Ahn* et al. only achieves the less common notion of selective unforgeability [1]. There exists many additional work on RSSs. However, most of the schemes presented are not private, e.g., [23], [28], following the privacy notion formalized by *Brzuska* et al. [8]. In those schemes, a verifier can deduce statements about the original message m, which contradicts the intention of RSSs [8], [32]. Our construction achieves full unforgeability and an advanced definition of privacy, namely unlinkability. Existing work on RSSs does not treat accountability in a formal sense, but notes that this a desirable property, e.g., [18], [33]. [33] introduces a first idea, but lacks formal proofs and complete security model. This paper addresses these shortcomings.

On the other hand, SSSs have been introduced by *Ateniese* et al. [2]. They allow a semi-trusted third party, named the sanitizer, to change *admissible* blocks of a signed message to arbitrary bitstrings. The basic security properties have then been formalized by *Brzuska* et al. [7]. These have been extended for unlinkability [10], [12], [20] and public accountability [11], [12]. The concept of grouping blocks has recently been introduced in [17]. Extensions like limiting-to-values [13], [27], trapdoor-SSSs [15] and multi-sanitizer environments [9], [14] have also been considered. Our definitions only cover single sanitizer environments; how to extend our schemes to multi-sanitizer environments is discussed in Sect. IV.

There exists much additional related work, e.g., zero-knowledge structure queries [21], [22]. The same references provide a good overview of additional related work.

II. Preliminaries and Building Blocks

For our construction, we require RSSs and SSSs. We therefore completely define each of them to be self-contained.

$\lambda \in \mathbb{N}$ denotes the security parameter. All algorithms implicitly take 1^λ as their first input. For a message $m = (m[1], \dots, m[\ell])$, where $m[i] \in \{0,1\}^*$, we call $m[i]$ a block, while $\ell \in \mathbb{N}$ denotes the number of blocks in a message m. $\mathcal{P}(S)$ denotes the power set of S. We call an algorithm efficient, if it runs in probabilistic polynomial time ($\mathcal{PPT}$) in λ. All algorithms may return an exception $\perp \notin \{0,1\}^*$.

A. Redactable Signature Schemes

The following definitions for RSSs are compiled from [8], [34]. We do not consider dynamic updates or merging [31], as we also want to achieve unlinkability.

Definition 1 (Redactable Signature Scheme (RSS)): An RSS consists of four efficient algorithms. Let RSS := (KG_{sig}, Sign, Redact, Verify), such that:

1) Key Generation: The signer generates a key pair, based on the security parameter λ:

$$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^\lambda)$$

2) Signing: The Sign algorithm takes as input a message m and sk_{sig}. It outputs a signature σ:

$$\sigma \leftarrow \mathsf{Sign}(m, sk_{\text{sig}})$$

3) Redacting: Algorithm Redact takes a message m, MOD, a signature σ, and pk_{sig}. It modifies the message m according to the modification instruction $\text{MOD} \in \mathcal{P}(\{1, 2, \dots, \ell\})$. MOD contains the indices for those blocks that shall be redacted. $m' \leftarrow \text{MOD}(m)$ means that m is modified according to MOD. Redact outputs $m' \leftarrow \text{MOD}(m)$ and σ':

$$(m', \sigma') \leftarrow \mathsf{Redact}(m, \text{MOD}, \sigma, pk_{\text{sig}})$$

4) Verification: Algorithm Verify outputs a decision $d \in \{1, 0\}$ verifying the correctness of a signature σ for a message m w.r.t. a public key pk_{sig}:

$$d \leftarrow \mathsf{Verify}(m, \sigma, pk_{\text{sig}})$$

We require the usual correctness properties to hold. The security definitions can be found in App. A; our definition of unlinkability is also new, but is achieved by every RSS which is completely context hiding and the adversary is allowed to generate the key pair [3].

B. Sanitizable Signature Schemes

The definitions for SSSs are compiled from [7], [11], [12].

Definition 2 (Sanitizable Signature Scheme): A SSS consists of seven efficient algorithms. Let SSS := (KG_{sig}, KG_{san}, Sign, Sanit, Verify, Proof, Judge), such that:

1) Key Generation: There is one key generation algorithm for the signer and one for the sanitizer. Both create a key pair; a private key and the corresponding public key, w.r.t. the security parameter λ:

$$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^\lambda), (pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KG}_{\text{san}}(1^\lambda)$$

2) Signing: The Sign algorithm takes as input a message m, sk_{sig}, pk_{san}, as well as a description ADM of the admissibly modifiable blocks. ADM contains the indices of the modifiable blocks, as well as the number ℓ of blocks in m. It outputs a signature σ:

$$\sigma \leftarrow \mathsf{Sign}(m, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})$$

3) Sanitizing: Algorithm Sanit takes a message m, modification instruction MOD, a signature σ, pk_{sig}, and sk_{san}. It modifies the message m according to the modification instruction MOD, which is a set containing pairs $(i, m[i]')$ for those blocks that shall be modified, meaning that $m[i]$ is replaced with $m[i]'$. Sanit calculates a new signature σ' for the modified message $m' \leftarrow \text{MOD}(m)$. Then, Sanit outputs m' and σ':

$$(m', \sigma') \leftarrow \mathsf{Sanit}(m, \text{MOD}, \sigma, pk_{\text{sig}}, sk_{\text{san}})$$

4) Verification: The Verify algorithm outputs a decision $d \in \{1, 0\}$ verifying the correctness of a signature σ for a message m w.r.t. the public keys pk_{sig} and pk_{san}:

$$d \leftarrow \mathsf{Verify}(m, \sigma, pk_{\text{sig}}, pk_{\text{san}})$$

5) Proof: The Proof algorithm takes as input sk_{sig}, a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0, 1\}^*$ and a signature σ and a set of (polynomially many) additional message/signature pairs $\{(m_i, \sigma_i)\}$ and pk_{san}. It outputs a string $\pi \in \{0, 1\}^*$:

$$\pi \leftarrow \mathsf{Proof}(sk_{\text{sig}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, pk_{\text{san}})$$

6) Judge: Algorithm Judge takes as input a message m, a signature σ, both public keys and a proof π. It outputs a decision $d \in \{\texttt{Sig}, \texttt{San}\}$ indicating whether the message/signature pair has been created by the signer or a sanitizer:

$$d \leftarrow \mathsf{Judge}(m, \sigma, pk_{\text{sig}}, pk_{\text{san}}, \pi)$$

As before, we require that all correctness properties hold.

III. Accountable Redactable Signatures Schemes

Accountable redactable signatures (ARSS) allow to derive the accountable party of a given signature/message pair (σ, m). Our security definitions are based on the ones given for SSSs to ease understanding of our goals. We define two different characteristics of accountability: an online form, where the original signer has to generate a proof, and an offline version where anyone is able to derive the accountable party (called publicly accountable). Also, our definition covers a single sanitizer only; an extension to multiple ones is beyond the scope of this paper, but we briefly discuss this possibility in Sect. IV.

Definition 3 (Accountable RSS *(*ARSS*)):* An ARSS consists of seven efficient algorithms. As ARSSs are an extension of RSSs, we need additional algorithms, while existing ones are altered. In particular, let $\mathsf{ARSS} := (\mathsf{KG}_{\text{sig}}, \mathsf{KG}_{\text{san}}, \mathsf{Sign}, \mathsf{Redact}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$, such that:

1) Key Generation: There are two key generation algorithms, one for the signer and one for the sanitizer. Both create a pair of keys, a private and the corresponding public key, w.r.t. the security parameter λ:

$$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^\lambda), (pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KG}_{\text{san}}(1^\lambda)$$

2) Signing: The Sign algorithm takes as input a message m, sk_{sig}, and pk_{san}. It outputs a signature σ:

$$\sigma \leftarrow \mathsf{Sign}(m, sk_{\text{sig}}, pk_{\text{san}})$$

3) Redacting: Algorithm Redact takes a message m, $\textsc{mod}$, a signature σ, pk_{sig}, and sk_{san}. $\textsc{mod}$ is defined as for RSSs. Then, Redact outputs m' and σ':

$$(m', \sigma') \leftarrow \mathsf{Redact}(m, \textsc{mod}, \sigma, pk_{\text{sig}}, sk_{\text{san}})$$

4) Verification: The Verify algorithm outputs a decision $d \in \{1, 0\}$ verifying the correctness of a signature σ for a message m w.r.t. the public keys pk_{sig} and pk_{san}:

$$d \leftarrow \mathsf{Verify}(m, \sigma, pk_{\text{sig}}, pk_{\text{san}})$$

```
Experiment $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathsf{ARSS}}(\lambda)$
  $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^\lambda)$
  $(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KG}_{\text{san}}(1^\lambda)$
  $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}}, \cdot), \mathsf{Redact}(\cdot,\cdot,\cdot,\cdot, sk_{\text{san}}), \mathsf{Proof}(sk_{\text{sig}}, \cdot,\cdot,\cdot,\cdot)}(pk_{\text{sig}}, pk_{\text{san}})$
    for $i = 1, 2, \ldots, q$ let $(m_i, pk_{\text{san},i})$ and $\sigma_i$
    index the queries/answers to/from $\mathsf{Sign}$
    for $j = 1, 2, \ldots, q'$ let $(m_j, \sigma_j, pk_{\text{sig},j}, \textsc{mod}_i)$ and $(m'_j, \sigma'_j)$
    index the queries/answers to/from $\mathsf{Redact}$
  return 1, if
    $\mathsf{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk_{\text{san}}) = 1$ and
    $\forall i \in \{1, 2, \ldots, q\} : (pk_{\text{san}}, m^*) \neq (pk_{\text{san},i}, m_i)$ and
    $\forall j \in \{1, 2, \ldots, q'\} : (pk_{\text{sig}}, m^*) \neq (pk_{\text{sig},j}, m'_j)$
```

Fig. 1. ARSS Unforgeability

5) Proof: The Proof algorithm takes as input sk_{sig}, a message m and a signature σ as well a set of (polynomially many) additional message/signature pairs $\{(m_i, \sigma_i)\}$ and pk_{san}. It outputs a string $\pi \in \{0, 1\}^*$:

$$\pi \leftarrow \mathsf{Proof}(sk_{\text{sig}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, pk_{\text{san}})$$

6) Judge: Algorithm Judge takes as input a message m, a signature σ, the public keys of both parties and a proof π. It outputs a decision $d \in \{\texttt{Sig}, \texttt{San}\}$ indicating whether the message/signature pair has been created by the signer or the sanitizer:

$$d \leftarrow \mathsf{Judge}(m, \sigma, pk_{\text{sig}}, pk_{\text{san}}, \pi)$$

Again, we require all correctness requirements to hold. As for SSSs [11], in a publicly accountable ARSS, we require that Judge outputs the accountable party with an empty proof ($\pi = \perp$).

A. *Security Model* ARSS

For ARSSs we require that a public key can efficiently be derived from its corresponding secret key. Let us explicitly stress that our definitions are similar to the ones for RSSs and SSSs; they blend in easily with the large body of existing work.

1) Unforgeability: The most basic requirement is unforgeability: no one should be able to generate valid signatures on new messages not issued before without having access to the private signing keys. This is captured within the next definition.

*Definition 4 (*ARSS *Unforgeability):* An ARSS is unforgeable, if for any efficient adversary $\mathcal{A}$ the probability that the game depicted in Fig. 1 returns 1, is negligible (as a function of λ). This is similar to standard signatures.

2) Sanitizer-Unforgeability: A sanitizer should not be able to alter a given message in a malicious way. In particular, a sanitizer must only redact, not append or alter a block or "exchange" the public key. $\text{span}_{\models}(m)$ denotes the set which contains all messages derivable from $m = (m[1], m[2], \ldots, m[\ell])$ exclusively using redactions. More formally, let $\mathcal{I}$ denote the index set of $\mathcal{P}(\{1, 2, \ldots, \ell\})$; let $\text{span}_{\models}(m) := \bigcup_{i \in \mathcal{I}} \textsc{mod}^i(m)$, where $\textsc{mod}^i$ corresponds to the ith element in $\mathcal{P}(\{1, 2, \ldots, \ell\})$. The definition is as follows.

*Definition 5 (*ARSS *Sanitizer Unforgeability):* An ARSS is sanitizer unforgeable, if for any efficient adversary $\mathcal{A}$ the probability that the game depicted in Fig. 2 returns 1, is negligible (as a function of λ).

```
**Experiment** $\mathsf{Sanitizer\text{-}Unforgeability}_{\mathcal{A}}^{\mathsf{ARSS}}(\lambda)$
    $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^{\lambda})$
    $(m^*, \sigma^*, pk^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}}, \cdot), \mathsf{Proof}(sk_{\text{sig}}, \cdot, \cdot, \cdot, \cdot)}(pk_{\text{sig}})$
        for $i = 1, 2, \ldots, q$ let $(m_i, pk_{\text{san},i})$ index the queries to $\mathsf{Sign}$.
    return 1, if
        $\mathcal{Q} \leftarrow \bigcup_{(m_i, pk^*) \text{ was queried to } \mathsf{Sign}} \text{span}_{\models}(m_i)$
        $\mathsf{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk^*) = 1$ and
        $m^* \notin \mathcal{Q}$
```

Fig. 2. ARSS Sanitizer-Unforgeability

```
**Experiment** $\mathsf{Privacy}_{\mathcal{A}}^{\mathsf{ARSS}}(\lambda)$
    $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^{\lambda})$
    $(pk_{\text{san}}, pk_{\text{san}}) \leftarrow \mathsf{KG}_{\text{san}}(1^{\lambda})$
    $b \leftarrow \{0, 1\}$
    $a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}}, \cdot), \mathsf{Redact}(\cdot, \cdot, \cdot, \cdot, sk_{\text{san}}), \mathsf{Proof}(sk_{\text{sig}}, \cdot, \cdot, \cdot, \cdot)}_{\mathsf{LoRRedact}(\cdot, \cdot, \cdot, \cdot, sk_{\text{sig}}, sk_{\text{san}}, b)}(pk_{\text{sig}}, pk_{\text{san}})$
        where oracle $\mathsf{LoRRedact}$ on input of $m_0, \text{MOD}_0, m_1, \text{MOD}_1$:
        if $\text{MOD}_0(m_0) \neq \text{MOD}_1(m_1)$, return $\perp$
        let $\sigma \leftarrow \mathsf{Sign}(m_b, sk_{\text{sig}}, pk_{\text{san}})$
        return $(m', \sigma') \leftarrow \mathsf{Redact}(m_b, \text{MOD}_b, \sigma, pk_{\text{sig}}, sk_{\text{san}})$
    return 1, if $a = b$
```

Fig. 3. ARSS Privacy

```
**Experiment** $\mathsf{Transparency}_{\mathcal{A}}^{\mathsf{ARSS}}(\lambda)$
    $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^{\lambda})$
    $(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KG}_{\text{san}}(1^{\lambda})$
    $b \leftarrow \{0, 1\}$
    $a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}}, \cdot), \mathsf{Redact}(\cdot, \cdot, \cdot, \cdot, sk_{\text{san}}), \mathsf{Proof}(sk_{\text{sig}}, \cdot, \cdot, \cdot, \cdot)}_{\mathsf{Redact/Sign}(\cdot, \cdot, sk_{\text{sig}}, sk_{\text{san}}, b)}(pk_{\text{sig}}, pk_{\text{san}})$
        where oracle $\mathsf{Redact/Sign}$ for input $m, \text{MOD}$
        $\sigma \leftarrow \mathsf{Sign}(m, sk_{\text{sig}}, pk_{\text{san}})$
        $(m', \sigma') \leftarrow \mathsf{Redact}(m, \text{MOD}, \sigma, pk_{\text{sig}}, sk_{\text{san}})$
        if $b = 1$: $\sigma' \leftarrow \mathsf{Sign}(m', sk_{\text{sig}}, pk_{\text{san}})$
        return $(m', \sigma')$
    if $\mathcal{A}$ has queried any message output by $\mathsf{Redact/Sign}$ to $\mathsf{Proof}$,
    return a random bit. Else, return 1, if $a = b$
```

Fig. 4. ARSS Transparency

```
**Experiment** $\mathsf{Unlinkability}_{\mathcal{A}}^{\mathsf{ARSS}}(\lambda)$
    $(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KG}_{\text{san}}(1^{\lambda})$
    $b \leftarrow \{0, 1\}$
    $a \leftarrow \mathcal{A}^{\mathsf{Redact}(\cdot, \cdot, \cdot, \cdot, sk_{\text{san}}), \mathsf{LoRRedact}(\cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, sk_{\text{san}}, b)}(pk_{\text{san}})$
      where oracle $\mathsf{LoRRedact}$ on input of:
      $m_0, \text{MOD}_0, \sigma_0, m_1, \text{MOD}_1, \sigma_1, pk_{\text{sig}}$
      if $\text{MOD}_0(m_0) \neq \text{MOD}_1(m_1)$, return $\perp$
      if $\mathsf{Verify}(m_0, \sigma_0, pk_{\text{sig}}, pk_{\text{san}}) \neq \mathsf{Verify}(m_1, \sigma_1, pk_{\text{sig}}, pk_{\text{san}})$, return $\perp$
      return $(m', \sigma') \leftarrow \mathsf{Redact}(m_b, \text{MOD}_b, \sigma_b, pk_{\text{sig}}, sk_{\text{san}})$
    return 1, if $a = b$
```

Fig. 5. ARSS Unlinkability

3) Privacy: A third party should not be able to derive any knowledge about redacted elements. This is covered within the next definition similar to the definitions for RSSs and SSSs.

*Definition 6 (*ARSS *Privacy):* An ARSS is *private*, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 3 returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ).

4) Transparency: An even stronger privacy definition is transparency. It prohibits finding the accountable party of given signature without knowing the signer's secret key, i.e., whether a signature was created by Sign or Redact. Note, access to the proof-oracle is limited.

*Definition 7 (*ARSS *Transparency):* An ARSS is proof-restricted transparent, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 4 returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ).

5) Unlinkability: In a nutshell, the adversary can input two signatures/message pairs. The LoRRedact-oracle chooses one of the two signatures to generate a new one. Here, the adversary has to guess which of the two inputted message-signature pairs was chosen to be sanitized. Note, compared to the privacy definition, also the signatures are input by the adversary, not internally generated. That is also the reason why we let the adversary specify each pk_{sig}, following [12], which formalized strong unlinkability for SSSs.

*Definition 8 (*ARSS *Unlinkability):* An ARSS is unlinkable, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 5 returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ).

6) Signer Accountability: In the following definition, the adversary has to generate a proof π^* which makes Judge to decide that the sanitizer is accountable, if it is not.

*Definition 9 (*ARSS *Signer Accountability):* An ARSS is signer accountable, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 6 returns 1 is negligible (as a function of λ).

7) Sanitizer Accountability: In the following game, the adversary has to generate a message/signature (m^*, σ^*) which makes Proof generate a proof π, leading the Judge to decide that the signer is accountable, while it is not.

*Definition 10 (*ARSS *Sanitizer Accountability):* A redactable signature scheme ARSS is sanitizer accountable, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 7 returns 1 is negligible (as a function of λ).

8) Public Accountability: The basic idea for defining public accountability is that an adversary, i.e., the sanitizer or the signer, has to be able to make the Judge decide wrongly on an empty proof π. This captures the idea that public accountability allows an outsider to derive the accountable party on its own: it is public as running Proof is not necessary, and Judge does not require any secret material.

*Definition 11 (*ARSS *Public Accountability):* A redactable signature scheme ARSS is publicly accountable, if $\mathsf{Proof} = \perp$, and, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 8 returns 1 is negligible (as a function of λ).

We call an ARSS secure, if it is correct, unlinkable, unforgeable, sanitizer-unforgeable, private, sanitizer-accountable and signer-accountable.

We do not consider transparency or public accountability as integral security requirements as they are mutually exclusive. Additionally, it depends on the use case which security notion is required.

```
Experiment Sig-Accountability$^{\mathsf{ARSS}}_{\mathcal{A}}(\lambda)$
    $(pk_{san}, sk_{san}) \leftarrow \mathsf{KG}_{san}(1^\lambda)$
    $b \leftarrow \{0,1\}$
    $(pk^*, \pi^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Redact}(\cdot,\cdot,\cdot,\cdot,sk_{san})}(pk_{san})$
        let $(m'_i, \sigma'_i)$ for $i = 1, \ldots, q$
        denote the answers from oracle Redact
    return 1, if
        $\mathsf{Verify}(m^*, \sigma^*, pk^*, pk_{san}) = 1$ and
        $\forall i \in \{1, 2, \ldots, q\} : (pk^*, m^*) \neq (pk_{sig,i}, m'_i)$ and
        $\mathsf{Judge}(m^*, \sigma^*, pk^*, pk_{san}, \pi^*) = \mathtt{San}$
```

Fig. 6. ARSS Signer Accountability

```
Experiment San-Accountability$^{\mathsf{ARSS}}_{\mathcal{A}}(\lambda)$
    $(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KG}_{sig}(1^\lambda)$
    $b \leftarrow \{0,1\}$
    $(pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,sk_{sig},\cdot),\mathsf{Proof}(sk_{sig},\cdot,\cdot,\cdot,\cdot)}(pk_{sig})$
        let $(m_i, pk_{san,i})$ and $\sigma_i$ for $i = 1, \ldots, q$
        denote the queries and answers to/from oracle Sign
    $\pi \leftarrow \mathsf{Proof}(sk_{sig}, m^*, \sigma^*, \{(m_i, \sigma_i) \mid 0 < i \leq q\}, pk^*)$
    return 1, if
        $\mathsf{Verify}(m^*, \sigma^*, pk_{sig}, pk^*) = 1$ and
        $\forall i \in \{1, 2, \ldots, q\} : (pk^*, m^*) \neq (pk_{san,i}, m_i)$ and
        $\mathsf{Judge}(m^*, \sigma^*, pk_{sig}, pk^*, \pi) = \mathtt{Sig}$
```

Fig. 7. ARSS Sanitizer Accountability

IV. Construction

In this section, we introduce the constructions for ARSSs. The basic idea is to use an SSS which makes any RSS accountable by signing the original message m and the corresponding redactable signature σ_{RSS} of that m using the SSS. While m and σ_{RSS} can be changed publicly by redaction using the RSS, the outer SSS signature can only be updated by dedicated and accountable sanitizers using the SSS. The construction is surprisingly simple; depending on the security properties of the underlying SSS, the resulting ARSS is either transparent or public accountable.

*Construction 1 (*ARSS*):* Provided secure RSS and SSS we define the accountable redactable signature ARSS = (KG_{sig}, KG_{san}, Sign, Redact, Verify, Proof, Judge) as follows:

1) Key Generation: KG_{sig} generates on input of the security parameter λ a key pair. It consists of a key pair for an RSS: $(pk^{\mathsf{RSS}}_{sig}, sk^{\mathsf{RSS}}_{sig}) \leftarrow \mathsf{RSS.KG}_{sig}(1^\lambda)$ and for an SSS: $(pk^{\mathsf{SSS}}_{sig}, sk^{\mathsf{SSS}}_{sig}) \leftarrow \mathsf{SSS.KG}_{sig}(1^\lambda)$. It returns $((pk^{\mathsf{SSS}}_{sig}, pk^{\mathsf{RSS}}_{sig}), (sk^{\mathsf{SSS}}_{sig}, sk^{\mathsf{RSS}}_{sig}))$. Analogously, KG_{san} returns a pair $(pk^{\mathsf{SSS}}_{san}, sk^{\mathsf{SSS}}_{san}) \leftarrow \mathsf{SSS.KG}_{san}(1^\lambda)$.
2) Signing: Sign, on input m, sk_{sig}, and pk_{san} computes $\sigma_{\mathsf{RSS}} \leftarrow \mathsf{RSS.Sign}(m, sk^{\mathsf{RSS}}_{sig})$. It lets $\sigma_{\mathsf{SSS}} \leftarrow \mathsf{SSS.Sign}((m, \sigma_{\mathsf{RSS}}, pk^{\mathsf{RSS}}_{sig}, pk^{\mathsf{SSS}}_{sig}, pk^{\mathsf{SSS}}_{san}), sk^{\mathsf{SSS}}_{sig}, pk^{\mathsf{SSS}}_{san}, \mathrm{ADM})$, where $\mathrm{ADM} = (\{1, 2\}, 5)$. It returns $\sigma = (\sigma_{\mathsf{RSS}}, \sigma_{\mathsf{SSS}})$.
3) Redacting: Redact, on input of message m, modification instructions MOD, a signature $\sigma = (\sigma_{\mathsf{RSS}}, \sigma_{\mathsf{SSS}})$, checks if σ is valid using Verify. If not, it returns $\perp$. If all checks pass, it generates $(m', \sigma'_{\mathsf{RSS}}) \leftarrow \mathsf{RSS.Redact}(m, \mathrm{MOD}, \sigma_{\mathsf{RSS}}, pk^{\mathsf{RSS}}_{sig})$ and $(m'', \sigma'_{\mathsf{SSS}}) \leftarrow \mathsf{SSS.Sanit}((m, \sigma_{\mathsf{RSS}}, pk^{\mathsf{RSS}}_{sig}, pk^{\mathsf{SSS}}_{sig}, pk^{\mathsf{SSS}}_{san}), \{(1, m'), (2, \sigma'_{\mathsf{RSS}})\}, \sigma_{\mathsf{SSS}}, pk^{\mathsf{SSS}}_{sig}, sk^{\mathsf{SSS}}_{san})$. It finally returns $(m', (\sigma'_{\mathsf{RSS}}, \sigma'_{\mathsf{SSS}}))$.
4) Verification: Algorithm Verify on input of a message m a signature $\sigma = (\sigma_{\mathsf{RSS}}, \sigma_{\mathsf{SSS}})$ and public keys pk_{sig} and pk_{san}, first checks if σ_{RSS} is a valid signature using $\mathsf{RSS.Verify}(m, \sigma_{\mathsf{RSS}}, pk^{\mathsf{RSS}}_{sig})$. It then checks whether σ_{SSS} is valid running $\mathsf{SSS.Verify}((m, \sigma_{\mathsf{RSS}}, pk^{\mathsf{RSS}}_{sig}, pk^{\mathsf{SSS}}_{sig}, pk^{\mathsf{SSS}}_{san}), \sigma_{\mathsf{SSS}}, pk^{\mathsf{SSS}}_{sig}, pk^{\mathsf{SSS}}_{san})$. If all checks are valid, return 1. Otherwise it returns 0.
5) Proof: Proof on input of $\sigma = (\sigma_{\mathsf{RSS}}, \sigma_{\mathsf{SSS}})$, m, pk_{san}, sk_{sig}, and $\{m_i, (\sigma_{\mathsf{RSS},i}, \sigma_{\mathsf{SSS},i}) \mid i \in \mathbb{N}\}$, it first checks whether σ is valid. If not, it returns $\perp$. Else, it returns $\pi \leftarrow \mathsf{SSS.Proof}(sk^{\mathsf{SSS}}_{sig}, (m, \sigma_{\mathsf{RSS}}, pk^{\mathsf{RSS}}_{sig}, pk^{\mathsf{SSS}}_{sig}, pk^{\mathsf{SSS}}_{san}), \sigma_{\mathsf{SSS}}, \{((m_i, \sigma_{\mathsf{RSS},i}, pk^{\mathsf{RSS}}_{sig}, pk^{\mathsf{SSS}}_{sig}, pk^{\mathsf{SSS}}_{san}), \sigma_{\mathsf{SSS},i}) \mid i \in \mathbb{N}\}, pk^{\mathsf{SSS}}_{san})$.
6) Judge: Judge on input of $m, \sigma = (\sigma_{\mathsf{RSS}}, \sigma_{\mathsf{SSS}})$, pk_{sig}, pk_{san} and π first checks whether σ is valid. If not, it outputs $\perp$. Else, it outputs $\mathsf{SSS.Judge}((m, \sigma_{\mathsf{RSS}}, pk^{\mathsf{RSS}}_{sig}, pk^{\mathsf{SSS}}_{sig}, pk^{\mathsf{SSS}}_{san}), \sigma_{\mathsf{SSS}}, pk^{\mathsf{SSS}}_{sig}, pk^{\mathsf{SSS}}_{san}, \pi)$.

```
Experiment Pubaccountability$^{\mathsf{ARSS}}_{\mathcal{A}}(\lambda)$
    $(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KG}_{sig}(1^\lambda)$
    $(pk_{san}, sk_{san}) \leftarrow \mathsf{KG}_{san}(1^\lambda)$
    $(pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,sk_{sig},\cdot,\cdot),\mathsf{Redact}(\cdot,\cdot,\cdot,\cdot,sk_{san})}(pk_{sig}, pk_{san})$
        Let $(m_i, \mathrm{ADM}_i, pk_{san,i})$ and $\sigma_i$ for $i = 1, 2, \ldots, q$
        be the queries and answers to and from oracle Sign
        return 1 if
            $\mathsf{Verify}(m^*, \sigma^*, pk_{sig}, pk^*) = 1$, and
            $\forall i \in \{1, 2, \ldots, q\} : (pk^*, m^*) \neq (pk_{san,i}, m_i)$ and
            $\mathsf{Judge}(m^*, \sigma^*, pk_{sig}, pk^*, \perp) = \mathtt{Sig}$
        Let $(m_j, \mathrm{MOD}_j, \sigma_j, pk_{sig,j})$ and $(m'_j, \sigma'_j)$ for $j = 1, 2, \ldots, q'$
        be the queries and answers to and from oracle Redact
        return 1 if
            $\mathsf{Verify}(m^*, \sigma^*, pk^*, pk_{san}) = 1$ and
            $\forall j \in \{1, 2, \ldots, q'\} : (pk^*, m^*) \neq (pk_{sig,j}, m'_j)$ and
            $\mathsf{Judge}(m^*, \sigma^*, pk^*, pk_{san}, \perp) = \mathtt{San}$
```

Fig. 8. ARSS Public Accountability

A. Correctness and Security of the Construction

The correctness can be derived by inspection.

Theorem 1 (Security of ARSS*):* If SSS is a secure sanitizable signature scheme, while RSS is a secure redactable signature scheme, the resulting ARSS is a secure accountable redactable signature scheme.

The proof can be found in App. C.

B. Multiple Sanitizers and More Complex Structures

To extend our construction to support multiple sanitizers [14], one uses an SSS which supports multiple sanitizers. Of course, the security model has to be adjusted accordingly.

As aforementioned, there are RSSs for more complex data-structures like trees and graphs. In our proofs, we do not use the list structure; we are therefore confident that our general construction carries over to different data-structures if the underlying RSS supports it. Of course, the algorithms and security definitions have to be adjusted accordingly. However, due to the generality of our construction this is only a diligent but routine piece of work.

V. Conclusion and Future Work

We have introduced the notion of accountable redactable signatures. We defined two types of accountability: (1) transparency meets stronger privacy guarantees, while (2) public accountable allows meeting current legal requirements [11], [25]. We derived a new construction combining any secure RSS with any secure SSS. It can be easily instantiated needing no additional building blocks. We leave it as future work to show the relations between the different security properties of ARSS and to derive a direct construction, mainly for efficiency.

VI. Acknowledgments

H. C. Pöhls was supported by the European Union's 7th Framework Programme (FP7) under grant agreement n° 609094 (RERUM) and Horizon 2020 Programme under grant agreement n° 644962 (PRISMACLOUD). K. Samelin was supported by the European Union's 7th Framework Programme (FP7) under grant agreement n° 318424 (FutureID).

References

[1] J. H. Ahn et al. 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*, pages 159–177. Springer, 2005.

[3] N. Attrapadung, B. Libert, and T. Peters. Computing on authenticated data: New privacy definitions and constructions. In *ASIACRYPT*, pages 367–385, 2012.

[4] N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *PKC*, pages 386–404, 2013.

[5] D. Bauer, D. M. Blough, and A. Mohan. Redactable signatures on data with dependencies and their application to personal health records. WPES '09, pages 91–100, 2009.

[6] J. Brown and D. M. Blough. Verifiable and redactable medical documents. In *AMIA 2012*, 2012.

[7] C. Brzuska et al. Security of Sanitizable Signatures Revisited. In *Proc. of PKC 2009*, pages 317–336. Springer, 2009.

[8] C. Brzuska et al. Redactable Signatures for Tree-Structured Data: Definitions and Constructions. ACNS'10, pages 87–104, 2010.

[9] C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Sanitizable signatures: How to partially delegate control for authenticated data. In *Proc. of BIOSIG*, volume 155 of *LNI*, pages 117–128. GI, 2009.

[10] C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of Sanitizable Signatures. In *PKC*, pages 444–461, 2010.

[11] C. Brzuska, H. C. Pöhls, and K. Samelin. Non-Interactive Public Accountability for Sanitizable Signatures. In *EuroPKI*, LNCS, 2012.

[12] C. Brzuska, H. C. Pöhls, and K. Samelin. Efficient and Perfectly Unlinkable Sanitizable Signatures without Group Signatures. In *EuroPKI*, volume 8341 of *LNCS*, pages 12–30. Springer-Verlag, 2013.

[13] S. Canard and A. Jambert. On extended sanitizable signature schemes. In *CT-RSA*, pages 179–194, 2010.

[14] S. Canard, A. Jambert, and R. Lescuyer. Sanitizable signatures with several signers and sanitizers. In *AFRICACRYPT*, pages 35–52, 2012.

[15] S. Canard, F. Laguillaumie, and M. Milhau. Trapdoor sanitizable signatures and their application to content protection. In *ACNS*, pages 258–276, 2008.

[16] E.-C. Chang, C. L. Lim, and J. Xu. Short Redactable Signatures Using Random Trees. CT-RSA '09, pages 133–147. Springer, 2009.

[17] H. de Meer, H. C. Pöhls, J. Posegga, and K. Samelin. Scope of security properties of sanitizable signatures revisited. In *ARES*, pages 188–197, 2013.

[18] H. de Meer, H. C. Pöhls, J. Posegga, and K. Samelin. Redactable signature schemes for trees with signer-controlled non-leaf-redactions. In *E-Business and Telecommunications*, volume 455 of *CCIS*, pages 155–171. 2014.

```
**Experiment** $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathsf{RSS}}(\lambda)$
    $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^\lambda)$
    $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}})}(pk_{\text{sig}})$
        for $i = 1, 2, \ldots, q$ let $m_i$ index the queries to $\mathsf{Sign}$
    return 1, if
        $\mathsf{Verify}(m^*, \sigma^*, pk_{\text{sig}}) = 1$ and
        $m^* \notin \bigcup_{i=1}^{q} \mathsf{span}_{\vDash}(m_i)$
```

Fig. 9. RSS Unforgeability

[19] D. Derler, C. Hanser, and D. Slamanig. Blank digital signatures: Optimization and practical experiences. In *Privacy and Identity Management for the Future Internet in the Age of Globalisation*, volume 457, pages 201–215. Springer International Publishing, 2014.

[20] N. Fleischhacker et al. Efficient unlinkable sanitizable signatures from signatures with rerandomizable keys. *ePrint*, 395, 2015.

[21] E. Ghosh, M. T. Goodrich, O. Ohrimenko, and R. Tamassia. Fully-dynamic verifiable zero-knowledge order queries for network data. *ePrint*, 2015:283, 2015.

[22] E. Ghosh, O. Ohrimenko, and R. Tamassia. Verifiable member and order queries on a list in zero-knowledge. *ePrint*, 2014:632, 2014.

[23] S. Haber et al. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In *ASIACCS*, pages 353–362, 2008.

[24] C. Hanser and D. Slamanig. Blank digital signatures. In *AsiaCCS*, pages 95 – 106. ACM, 2013.

[25] F. Höhne, H. C. Pöhls, and K. Samelin. Rechtsfolgen editierbarer signaturen. *Datenschutz und Datensicherheit*, 36(7):485–491, 2012.

[26] R. Johnson, D. Molnar, D. Song, and D.Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262. Springer, Feb. 2002.

[27] M. Klonowski and A. Lauks. Extended Sanitizable Signatures. In *ICISC*, pages 343–355, 2006.

[28] A. Kundu and E. Bertino. Privacy-preserving authentication of trees and graphs. *Int. J. Inf. Sec.*, 12(6):467–494, 2013.

[29] K. Miyazaki, S. Susaki, M. Iwamura, T. Matsumoto, R. Sasaki, and H. Yoshiura. Digital documents sanitizing problem, 2003.

[30] H. C. Pöhls, A. Bilzhause, K. Samelin, and J. Posegga. Sanitizable signed privacy preferences for social networks. In *DICCDI '11*, LNI. GI, October 2011.

[31] H. C. Pöhls and K. Samelin. On updatable redactable signatures. In *ACNS*, volume 8479 of *LNCS*, pages 457–475. Springer, 2014.

[32] H. C. Pöhls, K. Samelin, and J. Posegga. Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In *ACNS*, volume 6715 of *LNCS*, pages 166–182. Springer, 2011.

[33] K. Samelin et al. On Structural Signatures for Tree Data Structures. In *ACNS*, volume 7341 of *LNCS*, pages 171–187. Springer, 2012.

[34] K. Samelin et al. Redactable signatures for independent removal of structure and content. In *ISPEC*, volume 7232 of *LNCS*, pages 17–33. Springer, 2012.

[35] D. Slamanig and S. Rass. Generalizations and extensions of redactable signatures with applications to electronic healthcare. In *CMS*, pages 201–213, 2010.

[36] R. Steinfeld and L. Bull. Content extraction signatures. In *ICISC*. Springer Berlin / Heidelberg, 2001.

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

Appendix

A. RSS Security Model

1) Unforgeability: No one should be able to generate valid signatures on messages not endorsed by the signer without having access to sk_{sig}. Not endorsed excludes valid redactions from forgeries. This is captured within the following definition.

```
**Experiment** $\mathsf{Privacy}_{\mathcal{A}}^{\mathsf{RSS}}(\lambda)$
    $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^\lambda)$
    $b \leftarrow \{0,1\}$
    $a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}}), \mathsf{LoRRedact}(\cdot,\cdot,\cdot,\cdot, sk_{\text{sig}}, b)}(pk_{\text{sig}})$
        where oracle $\mathsf{LoRRedact}$ on input of $m_0, \text{MOD}_0, m_1, \text{MOD}_1$:
        if $\text{MOD}_0(m_0) \neq \text{MOD}_1(m_1)$, return $\perp$
        let $\sigma \leftarrow \mathsf{Sign}(m_b, sk_{\text{sig}})$
        return $(m', \sigma') \leftarrow \mathsf{Redact}(m_b, \text{MOD}_b, \sigma, pk_{\text{sig}})$
    return 1, if $a = b$
```

Fig. 10. RSS Privacy

```
**Experiment** $\mathsf{Transparency}_{\mathcal{A}}^{\mathsf{RSS}}(\lambda)$
    $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^\lambda)$
    $b \leftarrow \{0,1\}$
    $a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}}), \mathsf{Redact/Sign}(\cdot,\cdot, sk_{\text{sig}}, b)}(pk_{\text{sig}})$
        where oracle $\mathsf{Redact/Sign}$ for input $m, \text{MOD}$:
        $\sigma \leftarrow \mathsf{Sign}(m, sk_{\text{sig}})$
        $(m', \sigma') \leftarrow \mathsf{Redact}(m, \text{MOD}, \sigma, pk_{\text{sig}})$
        if $b = 1$: $\sigma' \leftarrow \mathsf{Sign}(m', sk_{\text{sig}})$
        return $(m', \sigma')$
    return 1, if $a = b$
```

Fig. 11. RSS Transparency

```
**Experiment** $\mathsf{Unlinkability}_{\mathcal{A}}^{\mathsf{RSS}}(\lambda)$
    $b \leftarrow \{0,1\}$
    $a \leftarrow \mathcal{A}^{\mathsf{LoRRedact}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,b)}()$
        where oracle $\mathsf{LoRRedact}$ on input of:
        $m_0, \text{MOD}_0, \sigma_0, m_1, \text{MOD}_1, \sigma_1, pk_{\text{sig}}$
        if $\text{MOD}_0(m_0) \neq \text{MOD}_1(m_1)$, return $\perp$
        if $\mathsf{Verify}(m_0, \sigma_0, pk_{\text{sig}}) \neq \mathsf{Verify}(m_1, \sigma_1, pk_{\text{sig}})$, return $\perp$
        return $(m', \sigma') \leftarrow \mathsf{Redact}(m_b, \text{MOD}_b, \sigma_b, pk_{\text{sig}})$
    return 1, if $a = b$
```

Fig. 12. RSS Unlinkability

```
**Experiment** $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathsf{SSS}}(\lambda)$
    $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^\lambda)$
    $(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KG}_{\text{san}}(1^\lambda)$
    $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}},\cdot,\cdot), \mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot, sk_{\text{san}}), \mathsf{Proof}(sk_{\text{sig}},\cdot,\cdot,\cdot,\cdot)}(pk_{\text{sig}}, pk_{\text{san}})$
        for $i = 1, 2, \ldots, q$ let $(m_i, pk_{\text{san},i}, \text{ADM}_i)$ and $\sigma_i$
        index the queries/answers to/from $\mathsf{Sign}$
        for $j = 1, 2, \ldots, q'$ let $(m_j, \sigma_j, pk_{\text{sig},j}, \text{MOD}_j)$ and $(m'_j, \sigma'_j)$
        index the queries/answers to/from $\mathsf{Sanit}$
    return 1, if
        $\mathsf{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk_{\text{san}}) = 1$ and
        $\forall i \in \{1, 2, \ldots, q\} : (pk_{\text{san}}, m^*) \neq (pk_{\text{san},i}, m_i)$ and
        $\forall j \in \{1, 2, \ldots, q'\} : (pk_{\text{sig}}, m^*) \neq (pk_{\text{sig},j}, m'_j)$
```

Fig. 13. SSS Unforgeability

*Definition 12 (*RSS *Unforgeability):* An RSS is unforgeable, if for any efficient adversary $\mathcal{A}$ the probability that the experiment depicted in Fig. 9 returns 1, is negligible (as a function of λ). This is similar to the definitions of unforgeability of standard signatures.

2) Privacy: A third party should not be able to derive any knowledge about redacted elements. This is covered within the privacy definition. Essentially, the adversary has to decide whether the first or second message of it supplied was redacted by the LoRRedact to yield the presented redacted message.

*Definition 13 (*RSS *Privacy):* An RSS is private, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 10 returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ).

3) Transparency: Transparency gives even stronger privacy guarantees. It prohibits deciding whether a signature was generated using Redact or Sign. Here, the adversary also chooses the message m to be signed and the target message.

*Definition 14 (*RSS *Transparency):* An RSS is transparent, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 11 returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ).

4) Unlinkability: Another strong privacy notion is unlinkability. It prohibits deciding whether a signature was derived from another known one.

*Definition 15 (*RSS *Unlinkability):* An RSS is unlinkable, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 12 returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ).

B. SSS *Security Model*

1) Unforgeability: As before for RSSs, no one should be able to generate valid signatures on new messages not queried before without having access to any private keys. However, compared to RSSs, we need to also consider the sanitizer.

*Definition 16 (*SSS *Unforgeability):* An SSS is unforgeable, if for any efficient adversary $\mathcal{A}$ the probability that the experiment depicted in Fig. 13 returns 1, is negligible (as a function of λ).

2) Immutability: A sanitizer must not be able to alter a given message in a malicious way. In particular, it must only be able to alter *admissible* blocks. Hence, also deleting or appending blocks must be prohibited. $\text{ADM}(\text{MOD}) = 1$ means that MOD only contains modifications which are admissible.

*Definition 17 (*SSS *Immutability):* An SSS is immutable, if for any efficient adversary $\mathcal{A}$ the probability that the experiment depicted in Fig. 14 returns 1, is negligible (as a function of λ).

3) Privacy: Privacy for SSSs is very related to the one of RSSs. In a nutshell, an adversary must not be able to derive knowledge about sanitized parts when it does not have access to them.

*Definition 18 (*SSS *Privacy):* An SSS is private, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 15 returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ). Here, the adversary has to decide which message was used to produce the desired outcome.

4) Unlinkability: In our unlinkability definition, the adversary has to decide which signature was used. This is the stronger notion from [12].

*Definition 19 (*SSS *Unlinkability):* An SSS is unlinkable, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 16 returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ). Here, the adversary has to guess which of the two message-signature pairs, supplied by the adversary, was chosen by the oracle LoRSanit to be sanitized.

```
Experiment $\mathsf{Immutability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
    $(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KG}_{\mathrm{sig}}(1^\lambda)$
    $(m^*, \sigma^*, pk^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\mathrm{sig}}, \cdot, \cdot), \mathsf{Proof}(sk_{\mathrm{sig}}, \cdot, \cdot, \cdot, \cdot)}(pk_{\mathrm{sig}})$
        for $i = 1, 2, \ldots, q$ let $(m_i, pk_{\mathrm{san},i}, \mathrm{ADM}_i)$ index the queries to oracle $\mathsf{Sign}$
    return 1, if
        $\mathsf{Verify}(m^*, \sigma^*, pk_{\mathrm{sig}}, pk^*) = 1$, and
        $\forall i \in \{1, 2, \ldots, q\} : pk^* \neq pk_{\mathrm{san},i}$ or $m^* \notin \{\mathrm{MOD}(m_i) \mid \mathrm{MOD}$ with $\mathrm{ADM}_i(\mathrm{MOD}) = 1\}$
```

Fig. 14. SSS Immutability

```
Experiment $\mathsf{Privacy}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
    $(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KG}_{\mathrm{sig}}(1^\lambda)$
    $(pk_{\mathrm{san}}, pk_{\mathrm{san}}) \leftarrow \mathsf{KG}_{\mathrm{san}}(1^\lambda)$
    $b \leftarrow \{0, 1\}$
    $a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\mathrm{sig}}, \cdot, \cdot), \mathsf{Sanit}(\cdot, \cdot, \cdot, \cdot, sk_{\mathrm{san}}), \mathsf{Proof}(sk_{\mathrm{sig}}, \cdot, \cdot, \cdot, \cdot)}_{\mathsf{LoRSanit}(\cdot, \cdot, \cdot, \cdot, \cdot, sk_{\mathrm{sig}}, sk_{\mathrm{san}}, b)}(pk_{\mathrm{sig}}, pk_{\mathrm{san}})$
        where oracle $\mathsf{LoRSanit}$ on input of:
        $m_0, \mathrm{MOD}_0, m_1, \mathrm{MOD}_1, \mathrm{ADM}$
        if $\mathrm{MOD}_0(m_0) \neq \mathrm{MOD}_1(m_1)$, return $\perp$
        let $\sigma \leftarrow \mathsf{Sign}(m_b, sk_{\mathrm{sig}}, pk_{\mathrm{san}}, \mathrm{ADM})$
        return $(m', \sigma') \leftarrow \mathsf{Sanit}(m_b, \mathrm{MOD}_b, \sigma, pk_{\mathrm{sig}}, sk_{\mathrm{san}})$
    return 1, if $a = b$
```

Fig. 15. SSS Privacy

```
Experiment $\mathsf{Unlinkability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
    $(pk_{\mathrm{san}}, sk_{\mathrm{san}}) \leftarrow \mathsf{KG}_{\mathrm{san}}(1^\lambda)$
    $b \leftarrow \{0, 1\}$
    $a \leftarrow \mathcal{A}^{\mathsf{Sanit}(\cdot, \cdot, \cdot, \cdot, sk_{\mathrm{san}}), \mathsf{LoRSanit}(\cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, sk_{\mathrm{san}}, b)}(pk_{\mathrm{san}})$
        where oracle $\mathsf{LoRSanit}$ on input of:
        $m_0, \mathrm{MOD}_0, \sigma_0, m_1, \mathrm{MOD}_1, \sigma_1, pk_{\mathrm{sig}}$
        if $\mathrm{ADM}_0 \neq \mathrm{ADM}_1$, or $\mathrm{MOD}_0(m_0) \neq \mathrm{MOD}_1(m_1)$, return $\perp$
        if $\mathsf{Verify}(m_0, \sigma_0, pk_{\mathrm{sig}}, pk_{\mathrm{san}}) \neq \mathsf{Verify}(m_1, \sigma_1, pk_{\mathrm{sig}}, pk_{\mathrm{san}})$, return $\perp$
        return $(m', \sigma') \leftarrow \mathsf{Sanit}(m_b, \mathrm{MOD}_b, \sigma_b, pk_{\mathrm{sig}}, sk_{\mathrm{san}})$
    return 1, if $a = b$
```

Fig. 16. SSS Unlinkability

```
Experiment $\mathsf{Transparency}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
    $(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KG}_{\mathrm{sig}}(1^\lambda)$
    $(pk_{\mathrm{san}}, sk_{\mathrm{san}}) \leftarrow \mathsf{KG}_{\mathrm{san}}(1^\lambda)$
    $b \leftarrow \{0, 1\}$
    $a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\mathrm{sig}}, \cdot, \cdot), \mathsf{Sanit}(\cdot, \cdot, \cdot, \cdot, sk_{\mathrm{san}}), \mathsf{Proof}(sk_{\mathrm{sig}}, \cdot, \cdot, \cdot, \cdot)}_{\mathsf{Sanit/Sign}(\cdot, \cdot, \cdot, sk_{\mathrm{sig}}, sk_{\mathrm{san}}, b)}(pk_{\mathrm{sig}}, pk_{\mathrm{san}})$
        where oracle $\mathsf{Sanit/Sign}$ for input $m, \mathrm{MOD}, \mathrm{ADM}$:
        $\sigma \leftarrow \mathsf{Sign}(m, sk_{\mathrm{sig}}, pk_{\mathrm{san}}, \mathrm{ADM})$
        $(m', \sigma') \leftarrow \mathsf{Sanit}(m, \mathrm{MOD}, \sigma, pk_{\mathrm{sig}}, sk_{\mathrm{san}})$
        if $b = 1$: $\sigma' \leftarrow \mathsf{Sign}(m', sk_{\mathrm{sig}}, pk_{\mathrm{san}}, \mathrm{ADM})$
        return $(m', \sigma')$
    if $\mathcal{A}$ has queried any message output by $\mathsf{Sanit/Sign}$ to $\mathsf{Proof}$,
    return a random bit. Else, return 1, if $a = b$
```

Fig. 17. SSS Transparency

5) Transparency: As for RSSs, transparency for SSS prohibits an adversary from deciding whether a signature originated from Sign or from Sanit. Note, access to the proof-oracle is limited.

*Definition 20 (*SSS *Transparency):* An SSS is proof-restricted transparent, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 17 returns 1 is negligibly close to $\frac{1}{2}$ (as a function of λ).

The basic idea is that the adversary is not able to decide whether it sees a fresh signature or one created by Sanit. Note, the access to the proof oracle is also limited.

6) Signer Accountability: For accountability, a signer should not be able to blame a sanitizer if the sanitizer is actually not responsible for a given message. In this game, the adversary has to generate a proof π^* which makes Judge

```
Experiment $\mathsf{Sig\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
    $(pk_{\mathrm{san}}, sk_{\mathrm{san}}) \leftarrow \mathsf{KG}_{\mathrm{san}}(1^\lambda)$
    $b \leftarrow \{0, 1\}$
    $(pk^*, \pi^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sanit}(\cdot, \cdot, \cdot, \cdot, sk_{\mathrm{san}})}(pk_{\mathrm{san}})$
        for $i = 1, \ldots, q$ let $(m'_i, \sigma'_i)$ and $(m_i, \mathrm{MOD}_i, \sigma_i, pk_{\mathrm{sig},i})$
        index the answers/queries from/to the oracle $\mathsf{Sanit}$
    return 1, if
        $\mathsf{Verify}(m^*, \sigma^*, pk^*, pk_{\mathrm{san}}) = 1$, and
        $\forall i \in \{1, 2, \ldots, q\} : (pk^*, m^*) \neq (pk_{\mathrm{sig},i}, m'_i)$ and
        $\mathsf{Judge}(m^*, \sigma^*, pk^*, pk_{\mathrm{san}}, \pi^*) = \mathtt{San}$
```

Fig. 18. SSS Signer Accountability

```
Experiment $\mathsf{San\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
    $(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KG}_{\mathrm{sig}}(1^\lambda)$
    $b \leftarrow \{0, 1\}$
    $(pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\mathrm{sig}}, \cdot, \cdot), \mathsf{Proof}(sk_{\mathrm{sig}}, \cdot, \cdot, \cdot, \cdot)}(pk_{\mathrm{sig}})$
        let $(m_i, \mathrm{ADM}_i, pk_{\mathrm{san},i})$ and $\sigma_i$ for $i = 1, \ldots, q$
        denote the queries/answers to/from oracle $\mathsf{Sign}$
    $\pi \leftarrow \mathsf{Proof}(sk_{\mathrm{sig}}, m^*, \sigma^*, \{(m_i, \sigma_i) \mid 0 < i \leq q\}, pk^*)$
    return 1, if
        $\mathsf{Verify}(m^*, \sigma^*, pk_{\mathrm{sig}}, pk^*) = 1$, and
        $\forall i \in \{1, 2, \ldots, q\} : (pk^*, m^*) \neq (pk_{\mathrm{san},i}, m_i)$ and
        $\mathsf{Judge}(m^*, \sigma^*, pk_{\mathrm{sig}}, pk^*, \pi) = \mathtt{Sig}$
```

Fig. 19. SSS Sanitizer Accountability

to decide that the sanitizer is accountable, if it is not.

*Definition 21 (*SSS *Signer Accountability):* An SSS is signer accountable, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 18 returns 1 is negligible (as a function of λ).

7) Sanitizer Accountability: To achieve accountability, a sanitizer should not be able to blame the signer if the sanitizer is responsible for a given message. In this game, the adversary has to generate a message/signature (m^*, σ^*) which makes Proof generate a proof π, leading Judge to decide that the signer is accountable, if it is not.

*Definition 22 (*SSS *Sanitizer Accountability):* A SSS is sanitizer accountable, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 19 returns 1 is negligible (as a function of λ).

8) Public Accountability: Basically, an adversary, i.e., the sanitizer or the signer, has to be able to make Judge decide wrongly on an empty proof π. As only Proof needs a secret key, this captures the required form of public accountability.

*Definition 23 (*SSS *Public Accountability):* An SSS is publicly accountable, if Proof $= \perp$, and, if for any efficient adversary $\mathcal{A}$ the probability that the experiment given in Fig. 20 returns 1 is negligible (as a function of λ).

Note again: we do neither treat transparency nor public accountability as an essential security requirement, as it depends on the use-case which security property is desired. Suitable instantiations include, e.g., [12], [20].

C. Proof of Theorem 1

We need to show that our construction is unforgeable, signer-accountable, sanitizer-accountable, unlinkable, sanitizer-unforgeable, private, and either transparent or publicly accountable. We prove each property on its own. Note,

```
Experiment $\mathsf{Pubaccountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
    $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KG}_{\text{sig}}(1^\lambda)$
    $(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KG}_{\text{san}}(1^\lambda)$
    $(pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}}, \cdot, \cdot), \mathsf{Sanit}(\cdot, \cdot, \cdot, \cdot, sk_{\text{san}})}(pk_{\text{sig}}, pk_{\text{san}})$
        for $i = 1, 2, \ldots, q$ let $(m_i, \text{ADM}_i, pk_{\text{san},i})$ and $\sigma_i$
        index the queries and answers to and from oracle Sign
        return 1 if
            $\mathsf{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk^*) = 1$, and
            $\forall i \in \{1, 2, \ldots, q\} : (pk^*, m^*) \neq (pk_{\text{san},i}, m_i)$ and
            $\mathsf{Judge}(m^*, \sigma^*, pk_{\text{sig}}, pk^*, \perp) = \texttt{Sig}$
        for $j = 1, 2, \ldots, q'$ let $(m_j, \text{MOD}_j, \sigma_j, pk_{\text{sig},j})$ and $(m'_j, \sigma'_j)$
        index the queries and answers to and from oracle Sanit
        return 1 if
            $\mathsf{Verify}(m^*, \sigma^*, pk^*, pk_{\text{san}}) = 1$, and
            $\forall j \in \{1, 2, \ldots, q'\} : (pk^*, m^*) \neq (pk_{\text{sig},j}, m'_j)$ and
            $\mathsf{Judge}(m^*, \sigma^*, pk^*, pk_{\text{san}}, \perp) = \texttt{San}$
```

Fig. 20. SSS Public Accountability

transparency and public accountability are mutually exclusive; it is required that for transparency the SSS and the RSS are transparent, while for public accountability, only the SSS needs to be publicly accountable.

1) Unforgeability: According to the definition, the only cases where the adversary $\mathcal{A}$ wins the game is when it comes up with a valid signature σ^* for a new message m^* which has not been signed under the given public keys. We can reduce this case to the unforgeability of the underlying SSS. Let $\mathcal{A}$ be an algorithm which breaks the unforgeability of ARSS. We can then construct an algorithm $\mathcal{B}$ which uses $\mathcal{A}$ to break the unforgeability of the SSS:

1) $\mathcal{B}$ creates a key pair of an RSS, i.e., $(pk^{\mathsf{RSS}}_{\text{sig}}, sk^{\mathsf{RSS}}_{\text{sig}}) \leftarrow \mathsf{RSS.KG}_{\text{sig}}(1^\lambda)$
2) $\mathcal{B}$ receives both public keys, i.e., $pk^{\mathsf{SSS}}_{\text{sig}}$ and $pk^{\mathsf{SSS}}_{\text{san}}$ of the SSS to forge.
3) $\mathcal{B}$ passes $pk^{\mathsf{SSS}}_{\text{sig}}$, $pk^{\mathsf{SSS}}_{\text{san}}$, and $pk^{\mathsf{RSS}}_{\text{sig}}$ to $\mathcal{A}$ as the ARSS's public keys.
4) For every query made, $\mathcal{B}$ uses its own oracles and keys to simulate $\mathcal{A}$'s environment.
5) Eventually, $\mathcal{A}$ returns a message/signature pair (m^*, σ^*). If this tuple verifies under the given public keys and has not been queried to the signing or redacting oracles, m^* is also a fresh message for the underlying SSS. $\mathcal{B}$ can therefore return $((m^*, \sigma^*_{\mathsf{RSS}}, pk^{\mathsf{RSS}}_{\text{sig}}, pk^{\mathsf{SSS}}_{\text{sig}}, pk^{\mathsf{SSS}}_{\text{san}}), \sigma^*_{\mathsf{SSS}})$, as its own forgery attempt. The probability that $\mathcal{B}$ wins the SSS unforgeability game is the same as $\mathcal{A}$'s, as the simulation is perfect and the message m^* is fresh.

2) Sanitizer-Unforgeability: Both cases where the adversary $\mathcal{A}$ wins the game require it to present a valid signature σ^* for a message m^* never been endorsed by the signer under the given pk^*. We provide a reduction for each case.

Case 1: If pk^* was never queried we derive an algorithm $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the immutability requirement of the underlying SSS. $\mathcal{B}$ proceeds as follows:

1) $\mathcal{B}$ creates a key pair of an RSS, i.e., $(pk^{\mathsf{RSS}}_{\text{sig}}, sk^{\mathsf{RSS}}_{\text{sig}}) \leftarrow \mathsf{RSS.KG}_{\text{sig}}(1^\lambda)$
2) $\mathcal{B}$ receives the signer's public keys, i.e., $pk^{\mathsf{SSS}}_{\text{sig}}$ of the SSS to forge.
3) $\mathcal{B}$ passes $pk^{\mathsf{SSS}}_{\text{sig}}$ and $pk^{\mathsf{RSS}}_{\text{sig}}$ to $\mathcal{A}$ as the ARSS's public keys.
4) For every query made, $\mathcal{B}$ uses its own oracles and keys to simulate $\mathcal{A}$'s environment.
5) Eventually, $\mathcal{A}$ returns a message/signature pair (m^*, σ^*, pk^*), where $\sigma^* = (\sigma^*_{\mathsf{RSS}}, \sigma^*_{\mathsf{SSS}})$. By assumption, pk^* is also a fresh public key of the sanitizer of the underlying SSS. If this tuple verifies, $\mathcal{B}$ can therefore return $(m', \sigma^*_{\mathsf{SSS}}, pk^*)$, as $\sigma^* = (\sigma^*_{\mathsf{RSS}}, \sigma^*_{\mathsf{SSS}})$ where $m' = (m^*, \sigma^*_{\mathsf{RSS}}, pk^{\mathsf{RSS}}_{\text{sig}}, pk^{\mathsf{SSS}}_{\text{sig}}, pk^*)$ as its own forgery attempt. The probability that $\mathcal{B}$ wins the SSS immutability game is therefore the same as $\mathcal{A}$'s advantage of winning the sanitizer-unforgeability game of the ARSS, as the simulation is perfect.

Case 2: When a signed m^* should not have been derivable by redaction, i.e., $m^* \notin \bigcup_{i=1}^{q} \mathsf{span}_{\models}(m_i)$ for every query to the signing oracle with pk^*, we derive an algorithm $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the unforgeability requirement of the underlying RSS. $\mathcal{B}$ proceeds as follows.

1) $\mathcal{B}$ creates a key pair of an SSS, i.e., $(pk^{\mathsf{SSS}}_{\text{sig}}, sk^{\mathsf{SSS}}_{\text{sig}}) \leftarrow \mathsf{SSS.KG}_{\text{sig}}(1^\lambda)$
2) $\mathcal{B}$ receives the public keys, i.e., $pk^{\mathsf{RSS}}_{\text{sig}}$ of the RSS to forge.
3) $\mathcal{B}$ passes $pk^{\mathsf{SSS}}_{\text{sig}}$ and $pk^{\mathsf{RSS}}_{\text{sig}}$ to $\mathcal{A}$ as ARSS's public keys.
4) For every query made, $\mathcal{B}$ uses its own oracles and keys to simulate $\mathcal{A}$'s environment.
5) Eventually, $\mathcal{A}$ returns (m^*, σ^*, pk^*), where $\sigma^* = (\sigma^*_{\mathsf{RSS}}, \sigma^*_{\mathsf{SSS}})$. This tuple verifies, and, by assumption, m^* is not derivable by any query made. Hence, m^* is also a forgery of the RSS. $\mathcal{B}$ can therefore return $(m^*, \sigma^*_{\mathsf{RSS}})$ as its own forgery attempt. The probability that $\mathcal{B}$ wins the RSS unforgeability game is therefore the same as $\mathcal{A}$'s advantage of winning the sanitizer-unforgeability game of the ARSS.

3) Privacy: According to the definition of privacy, the adversary $\mathcal{A}$ has to guess a bit. We can simply use $\mathcal{A}$ to either break privacy of the SSS or the RSS. $\mathcal{B}$ uses $\mathcal{A}$ as a block-box:

1) $\mathcal{B}$ tosses a coin $b \leftarrow \{0, 1\}$.
2) If $b = 0$, $\mathcal{B}$ creates both key pairs of an SSS, i.e., $(pk^{\mathsf{SSS}}_{\text{sig}}, sk^{\mathsf{SSS}}_{\text{sig}}) \leftarrow \mathsf{SSS.KG}_{\text{sig}}(1^\lambda)$ and $(pk^{\mathsf{SSS}}_{\text{san}}, sk^{\mathsf{SSS}}_{\text{san}}) \leftarrow \mathsf{SSS.KG}_{\text{san}}(1^\lambda)$ and receives RSS's public key, $pk^{\mathsf{RSS}}_{\text{sig}}$.
3) If $b = 1$, $\mathcal{B}$ creates a key pair of an RSS, i.e., $(pk^{\mathsf{RSS}}_{\text{sig}}, sk^{\mathsf{RSS}}_{\text{sig}}) \leftarrow \mathsf{RSS.KG}_{\text{sig}}(1^\lambda)$ and receives SSS's public keys, i.e., $pk^{\mathsf{SSS}}_{\text{san}}$ and $pk^{\mathsf{SSS}}_{\text{sig}}$.
4) $\mathcal{B}$ passes $pk^{\mathsf{SSS}}_{\text{sig}}$, $pk^{\mathsf{SSS}}_{\text{san}}$ and $pk^{\mathsf{RSS}}_{\text{sig}}$ to $\mathcal{A}$ as the ARSS's public keys.
5) For every query made, $\mathcal{B}$ uses its own oracles/keys to simulate $\mathcal{A}$'s environment.
6) Eventually, $\mathcal{A}$ returns its own guess b^*. $\mathcal{B}$ returns b^* as its own guess. What is the probability that $\mathcal{B}$'s guess is correct? Assuming that $\mathcal{A}$'s advantage against privacy is ϵ, while at least one of the underlying building blocks must be non-private (we do not introduce additional information), $\mathcal{B}$'s advantage against privacy of either the RSS or the SSS is $\geq \frac{\epsilon}{2}$:

the simulation for the correct case was done with probability of $\frac{1}{2}$.

4) Unlinkability: In the definition of unlinkability, our adversary $\mathcal{A}$ has to guess a bit. Again, we can use $\mathcal{A}$ to either break unlinkability of the SSS or the RSS. As usual, we use $\mathcal{A}$ as a black-box in an algorithm $\mathcal{B}$:

1) $\mathcal{B}$ tosses a coin $b \leftarrow \{0,1\}$.
2) If $b = 0$, $\mathcal{B}$ creates a sanitizer key pair of an SSS, i.e., $(pk_{\text{san}}^{\text{SSS}}, sk_{\text{san}}^{\text{SSS}}) \leftarrow \text{SSS.KG}_{\text{san}}(1^\lambda)$.
3) If $b = 1$, $\mathcal{B}$ receives $(pk_{\text{san}}^{\text{SSS}})$ from its own challenger.
4) $\mathcal{B}$ passes $pk_{\text{san}}^{\text{SSS}}$ to $\mathcal{A}$ as the sanitizer's public key.
5) For every query made, $\mathcal{B}$ uses its own oracles/keys to simulate $\mathcal{A}$'s environment accordingly.
6) Eventually, $\mathcal{A}$ returns its guess b^*. $\mathcal{B}$ returns b^* as its own guess. What is the probability that $\mathcal{B}$'s guess is correct? Assuming that $\mathcal{A}$'s advantage against unlinkability is ϵ and not negligible, while at least one of the underlying building blocks must be non-unlinkability (we do not introduce additional information), $\mathcal{B}$'s advantage against unlinkability of either the RSS or the SSS is $\geq \frac{\epsilon}{2}$, as the simulation for the correct case was done with probability of exactly $\frac{1}{2}$.

5) Transparency: For transparency, the adversary $\mathcal{A}$ has to also guess a bit. We can simply use $\mathcal{A}$ to either break transparency of the SSS or the RSS. Once more, we use $\mathcal{A}$ as a black-box in an algorithm $\mathcal{B}$:

1) $\mathcal{B}$ tosses a coin $b \leftarrow \{0,1\}$.
2) If $b = 0$, $\mathcal{B}$ creates both key pairs of an SSS, i.e., $(pk_{\text{sig}}^{\text{SSS}}, sk_{\text{sig}}^{\text{SSS}}) \leftarrow \text{SSS.KG}_{\text{sig}}(1^\lambda)$ and $(pk_{\text{san}}^{\text{SSS}}, sk_{\text{san}}^{\text{SSS}}) \leftarrow \text{SSS.KG}_{\text{san}}(1^\lambda)$ and receives RSS's public key, $pk_{\text{sig}}^{\text{RSS}}$.
3) If $b = 1$, $\mathcal{B}$ creates a key pair of an RSS, i.e., $(pk_{\text{sig}}^{\text{RSS}}, sk_{\text{sig}}^{\text{RSS}}) \leftarrow \text{RSS.KG}_{\text{sig}}(1^\lambda)$ and receives the public keys of the SSS, i.e., $pk_{\text{san}}^{\text{SSS}}$ and $pk_{\text{sig}}^{\text{SSS}}$.
4) $\mathcal{B}$ passes $pk_{\text{sig}}^{\text{SSS}}$, $pk_{\text{san}}^{\text{SSS}}$ and $pk_{\text{sig}}^{\text{RSS}}$ to $\mathcal{A}$ as the ARSS's public keys.
5) For every query made, $\mathcal{B}$ uses its own oracles/keys to simulate $\mathcal{A}$'s environment.
6) Eventually, $\mathcal{A}$ returns its own guess b^*. $\mathcal{B}$ returns b^* as its own guess. Assuming that $\mathcal{A}$'s advantage against transparency is ϵ, while at least one of the underlying building blocks must be non-transparent (once more: we do not introduce additional information in our construction), $\mathcal{B}$'s advantage against transparency of either the RSS or the SSS is $\geq \frac{\epsilon}{2}$, as the simulation for the correct case was done with probability $\frac{1}{2}$.

6) Signer-Accountability: According to the definition of signer-accountability, the adversary $\mathcal{A}$ has to generate $pk^*, \pi^*, m^*, \sigma^*$ which makes Judge decide wrongly, i.e., it outputs `San` while (m^*, pk^*) has never been queried to Redact. We use $\mathcal{A}$ as a black-box in an algorithm $\mathcal{B}$ to break the signer-accountability of the underlying SSS:

1) $\mathcal{B}$ receives $pk_{\text{san}}^{\text{SSS}}$ from its own challenger.
2) $\mathcal{B}$ passes $pk_{\text{san}}^{\text{SSS}}$ to $\mathcal{A}$ as the sanitizer's public key.
3) For every query made, $\mathcal{B}$ uses its own Sanit-oracle to simulate $\mathcal{A}$'s environment.
4) Eventually, $\mathcal{A}$ returns $(pk^*, \pi^*, m^*, \sigma^*)$. $\mathcal{B}$ returns $(pk_{\text{sig}}^{\text{SSS}*}, \pi^*, (m^*, \sigma_{\text{RSS}}^*, pk_{\text{sig}}^{\text{RSS}*}, pk_{\text{sig}}^{\text{SSS}*}, pk_{\text{san}}^{\text{SSS}}), \sigma_{\text{SSS}}^*)$, where $\sigma^* = (\sigma_{\text{RSS}}^*, \sigma_{\text{SSS}}^*)$ and $pk^* = (pk_{\text{sig}}^{\text{SSS}*}, pk_{\text{sig}}^{\text{RSS}*})$. As we only forward queries, and either m^* or pk^* are "fresh", the probability that $\mathcal{B}$ wins the SSS signer-accountability game is therefore the same as $\mathcal{A}$'s advantage of winning the signer-unforgeability game of the ARSS.

7) Sanitizer-Accountability: The adversary $\mathcal{A}$ has to generate pk^*, m^*, σ^* which makes Proof output a proof π which makes Judge decide wrongly, i.e., Judge outputs `Sig` while (m^*, pk^*) has never been queried to Sign. We use $\mathcal{A}$ as a black-box in an algorithm $\mathcal{B}$ again to break the sanitizer-accountability of the underlying SSS:

1) $\mathcal{B}$ creates a key pair of an RSS, i.e., $(pk_{\text{sig}}^{\text{RSS}}, sk_{\text{sig}}^{\text{RSS}}) \leftarrow \text{RSS.KG}_{\text{sig}}(1^\lambda)$
2) $\mathcal{B}$ receives $pk_{\text{sig}}^{\text{SSS}}$ from its own challenger.
3) $\mathcal{B}$ passes $pk_{\text{sig}}^{\text{SSS}}$ and $pk_{\text{sig}}^{\text{RSS}}$ to $\mathcal{A}$ as the signer's public keys.
4) For every query made, $\mathcal{B}$ uses its own oracles and keys to simulate $\mathcal{A}$'s environment.
5) Eventually, $\mathcal{A}$ returns (pk^*, m^*, σ^*). $\mathcal{B}$ returns $(pk_{\text{san}}^*, (m^*, \sigma_{\text{RSS}}^*, pk_{\text{sig}}^{\text{RSS}}, pk_{\text{sig}}^{\text{SSS}}, pk^*), \sigma_{\text{SSS}}^*)$, where $\sigma^* = (\sigma_{\text{RSS}}^*, \sigma_{\text{SSS}}^*)$. As we only forward queries, and either m^* or pk^* are "fresh", the probability that $\mathcal{B}$ wins the SSS sanitizer-accountability game is therefore the same as $\mathcal{A}$'s advantage of winning the sanitizer-unforgeability game of the ARSS.

8) Public Accountability: According to the definition of public accountability, the adversary $\mathcal{A}$ has to generate pk^*, m^*, σ^* which makes Judge decide wrongly on an empty proof $\pi = \perp$, or the adversary came up with a new public key/message pair not queried. In both cases, we can reduce the security of ARSS to the one of the underlying SSS. In particular, we use $\mathcal{A}$ as a black-box in an algorithm $\mathcal{B}$ again to break the public accountability of the underlying SSS:

1) $\mathcal{B}$ creates RSS key pair $(pk_{\text{sig}}^{\text{RSS}}, sk_{\text{sig}}^{\text{RSS}}) \leftarrow \text{RSS.KG}_{\text{sig}}(1^\lambda)$
2) $\mathcal{B}$ receives $(pk_{\text{sig}}^{\text{SSS}}, pk_{\text{san}}^{\text{SSS}})$ from its own challenger.
3) $\mathcal{B}$ forwards public keys to $\mathcal{A}$ as the ARSS's public keys.
4) For every query made, $\mathcal{B}$ uses its own oracles/keys to simulate $\mathcal{A}$'s environment.
5) Eventually, $\mathcal{A}$ returns (pk^*, m^*, σ^*). If pk^* only consists of one value, $\mathcal{B}$ returns $(pk^*, (m^*, \sigma_{\text{RSS}}^*, pk_{\text{sig}}^{\text{RSS}}, pk_{\text{sig}}^{\text{SSS}}, pk^*), \sigma_{\text{SSS}}^*)$, where $\sigma^* = (\sigma_{\text{RSS}}^*, \sigma_{\text{SSS}}^*)$ and $pk^* = pk_{\text{san}}^{\text{SSS}*}$. If pk^* consists of two public keys, $\mathcal{B}$ returns $(pk_{\text{sig}}^{\text{SSS}*}, (m^*, \sigma_{\text{RSS}}^*, pk_{\text{sig}}^{\text{RSS}*}, pk_{\text{sig}}^{\text{SSS}*}, pk_{\text{san}}^{\text{SSS}}), \sigma_{\text{SSS}}^*)$, where $\sigma^* = (\sigma_{\text{RSS}}^*, \sigma_{\text{SSS}}^*)$ and $pk^* = (pk_{\text{sig}}^{\text{RSS}*}, pk_{\text{sig}}^{\text{SSS}*})$. What is the probability that $\mathcal{B}$ wins its own game? If pk^* was never queried, but all public keys are signed by the SSS, the public accountability of the underlying SSS is obviously broken. In case Judge decides wrongly, then m^* was never queried to the corresponding oracle, also winning the public accountability of the underlying SSS. Hence, $\mathcal{B}$'s advantage equals the one of $\mathcal{A}$.