This is the full version of a paper which appears in CANS 2018,
17th International Conference on Cryptology And Network Security, Naples, Italy, Sept 30 - Oct 3, 2018,
Proceedings. LNCS 11124. pages 256–276. © Springer, 2018.
The final authenticated version is available online at https://doi.org/10.1007/978-3-030-00434-7_13

Protean Signature Schemes[‡]

Stephan Krenn[1], Henrich C. Pöhls[2], Kai Samelin[3‖], and Daniel Slamanig[1]

[1] AIT Austrian Institute of Technology, Vienna, Austria
{stephan.krenn,daniel.slamanig}@ait.ac.at
[2] ISL & Chair of IT-Security, University of Passau, Passau, Germany
hp@sec.uni-passau.de
[3] TÜV Rheinland i-sec GmbH, Hallbergmoos, Germany
kaispapers@gmail.com

Abstract. We introduce the notion of Protean Signature schemes. This novel type of signature scheme allows to remove and edit signer-chosen parts of signed messages by a semi-trusted third party simultaneously. In existing work, one is either allowed to remove or edit parts of signed messages, but not both at the same time. Which and how parts of the signed messages can be modified is chosen by the signer. Thus, our new primitive generalizes both redactable (Steinfeld et al., ICISC '01, Johnson et al., CT-RSA '02 & Brzuska et al., ACNS '10) and sanitizable signatures schemes (Ateniese et al., ESORICS '05 & Brzuska et al., PKC '09). We showcase a scenario where either primitive alone is not sufficient. Our provably secure construction (offering both strong notions of transparency and invisibility) makes only black-box access to sanitizable and redactable signature schemes, which can be considered standard tools nowadays. Finally, we have implemented our scheme; Our evaluation shows that the performance is reasonable.

1 Introduction

Standard unforgeable digital signature schemes do not allow for any alterations of signed messages, i.e., an adversary cannot generate validating signatures for messages not explicitly endorsed by the signer [27]. However, this is too limiting in many real-life scenarios. The standard use-case usually given as an example to clarify this situation is the handling of patient data. In particular, assume that a medical doctor signs a complete record consisting of the patient's name, insurance number and the received treatments. After the patient is released, the hospital's accountant receives the complete signed record related to the patient in question to prepare the bill for the insurance company.

Obviously, this is not very privacy-friendly, especially from the patient's point of view, as the accountant receives all information, even though some data is irrelevant for the task carried out. So, the obvious solution is to only give the treatments and the insurance number to the accountant, effectively anonymizing the paperwork. However, as standard signatures do not allow for such alterations, the medical doctor either needs to re-sign the document in this case or an additional trusted entity does it for the doctor. Still, both solutions are not very satisfactory, as it induces additional overhead and may also be impossible in certain scenarios, e.g., if the medical doctor is no longer employed. Thus, modifying signed messages in a controlled way has its merits.

Motivation and Contribution. We introduce the notion of "Protean Signature schemes" (PS). In such schemes, parts of a message can be removed, while the rest of this message remains authenticated, i.e., can be verified using the corresponding public key pk. However, in contrast to redactable signature schemes [32, 42] (RSS), which only allow removal of parts, our primitive also allows to alter signer-chosen blocks to arbitrary bit-strings, much like sanitizable signature schemes [2] (SSS), while also adding accountability.

[‡] This research was supported by European Union's Horizon 2020 research and innovation programme under grant agreement No 644962 PRISMACLOUD, No 653454 CREDENTIAL, No 783119 SECREDAS and No 321310 PERCY.

[‖] Part of this work was done while the third author was also at IBM Research – Zurich and TU Darmstadt.

In the above (minimal) example, removal is enough to anonymize the data in question. However, in some scenarios, it is also necessary to edit some parts of a signed message and not only to remove parts. For example, to achieve k-anonymity [43], one may want to coarsen (generalize) the ZIP-code by replacing the last digits of it with a special symbol, e.g., $*$. However, a problem here is that for more complex data, i.e., beyond ZIP-codes, the decision of which and how data may be anonymized depends on the actual data gathered and additional knowledge. Thus, there are scenarios where this information – and therefore the knowledge what to redact – is not available to the entity generating the signature and hence not known at the time of signature generation. Namely, coming back to the use-case with the patient data, data may be grouped based on side-effects on the type of medication given, while certain information must be completely removed, i.e., identifiers such as names. So, as new medications enter the market almost on a daily basis, even if based on existing agents, how should the signer be aware of potential side-effects or new treatments ahead of time, especially considering that the data must be grouped and anonymized w.r.t. to these values? Thus, a coexistence of the possibility to remove and edit signed data becomes a necessity to successfully protect privacy while maintaining data quality in certain scenarios.

This paper tackles this situation by introducing Protean Signature schemes (PS), where a semi-trusted third party can remove and edit signer-chosen parts of the message (blocks), also answering an open question by Bilzhause et al. [5] and de Meer et al. [19]. Thus, our new primitive can be seen as a generalization of both SSSs and RSSs.

In more detail, we introduce a formal security framework for PSs and a provably secure construction. Our construction is based on existing work on SSSs and RSSs, combining both concepts. The corresponding efficiency analysis and implementation shows that our construction is reasonably efficient, especially considering its possibilities.

Related Work. Malleable signatures received a lot of attention in the recent past, as it became clear that there are many application scenarios where signed messages need to be modified in a controlled way [1,5,20,25]. This weakens the standard unforgeability definition, where the messages protected by signatures cannot be altered at all, which is clearly not avoidable, if one wants to allow for modifications or derivations.

Essentially, existing work can be grouped into three, sometimes overlapping, directions. The first direction are homomorphic signatures [1, 6, 32, 44], and some other closely related concepts [7, 45]. Homomorphic signatures take several (signed) messages as input and can be used to compute functions on authenticated data-sets. Here, an entity not holding any secrets can derive a new (valid) signature σ' on $f(m)$, where the function f is public.

Directly related are RSSs, where anyone (i.e., no secrets are involved) can publish a subset of signed data, along with a new signature σ'. To illustrate this, let $m =$ (I, do, not, like, fish) along with a valid redactable signature σ. Anyone can then derive a signature σ' on $m' =$ (I, like, fish), i.e., redact the second and third block $m^2 =$ do and $m^3 =$ not. The original ideas of RSSs [32, 42] were later formalized [8, 35]. Then, RSSs have been extended to allow for additional use-cases, including adding accountability [38], discussing their relation to SSSs [19], allowing for redactable structure [40], prohibiting additional redactions [29–31, 37], yet also defining dependencies between different parts of a message [41]. Moreover, there are also some real-world implementations of this primitive proving that they are practical [39, 46]. All these approaches (but accountability) have later been unified into a generalized framework by Derler et al. [21]. Note, the work by Izu et al. [30] addresses the case of "sanitizable and deletable signatures". However, they actually address the case of RSSs and not SSSs. In particular, in their scheme, a third party can decide whether a redaction is visible or not, but does not allow for any other alterations. We follow the nomenclature clarified by Bilzhause et al. [5] and thus classify the work by Izu et al. [30] as an RSS.

Likewise, SSSs allow to alter signer-chosen blocks of signed messages by a semi-trusted entity named the sanitizer [2]. In particular, the sanitizer holds its own secret key and can derive new messages, along with the corresponding signatures, but cannot completely redact blocks. For example, if $m =$ (I, do, not, like, fish)

(and m^5 is admissible, i.e., modifiable), then the sanitizer can, e.g., derive a new signature σ' on the message $m' = (\text{I, do, not, like, meat})$. Even though this seems to be off the limits, it turned out that this primitive has many real-life application scenarios, see, e.g., Bilzhause et al. [5]. After the initial ideas by Ateniese et al. [2], SSSs also received a lot of attention in the recent past. Namely, the first thorough security model was given by Brzuska et al. [9] (later slightly modified by Gong et al. [28]), which was later extended for multiple signers/sanitizers [10, 16], unlinkability (which means a derived signatures cannot be linked to its original) [11, 13, 24, 36], trapdoor SSSs (where a signer can choose additional sanitizers after signature generation) [17,47], non-interactive public-accountability (an outsider can determine which party is accountable for a given valid message/signature pair) [12], limiting the sanitizer to signer-chosen values [15,22,33], invisibility (meaning that an outsider cannot derive which parts of a message are sanitizable) [3, 14, 23] and the case of strongly unforgeable signatures [34]. All these extensions allow for additional use-cases of this primitive [5].

Additional related work is given in some recent surveys [5, 20, 26]. We stress that a slightly altered SSS can be used to "mimic" an RSS by defining a special symbol to which the specific block is sanitized to mark the block as "redacted". However, as shown by de Meer et al. [19], this has a negative impact on the privacy guarantees of the resulting scheme because the special symbol remains visible. For example, $m' = (\text{I, like, fish})$ is clearly different from $m' = (\text{I}, \perp, \perp, \text{ like, fish})$. We stress that our scheme supports both possibilities, i.e., visible and non-visible (transparent) redactions, adding additional freedom for the signer.

2 Preliminaries and Notation

We now give our notation and the required preliminaries. These include labeled IND-CCA2 secure encryption schemes, sanitizable signatures and redactable signatures.

The formal definitions can be found in App. A.

Notation. The main security parameter is denoted by $\kappa \in \mathbb{N}$. All algorithms implicitly take 1^κ as an additional input. We write $a \leftarrow A(x)$ if a is assigned to the output of the deterministic algorithm A with input x. If an algorithm A is probabilistic, we write $a \leftarrow_r A(x)$. An algorithm is efficient if it runs in probabilistic polynomial time (PPT) in the length of its input. For the remainder of this paper, all algorithms are PPT if not explicitly mentioned otherwise. Most algorithms may return a special error symbol $\perp \notin \{0,1\}^*$, denoting an exception. If S is a set, we write $a \leftarrow_r S$ to denote that a is chosen uniformly at random from S. In the definitions, we speak of a general message space $\mathcal{M}$ to be as generic as possible. What $\mathcal{M}$ is concretely, is defined in the instantiations. For a message $m = (m^1, m^2, \ldots, m^{\ell_m})$, m^i is called a block and $\ell_m \in \mathbb{N}$ denotes the number of blocks in m. If m is clear from the context, it is dropped from ℓ_m. A function $\nu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is negligible, if it vanishes faster than every inverse polynomial, i.e., $\forall k \in \mathbb{N}$, $\exists n_0 \in \mathbb{N}$ such that $\nu(n) \leq n^{-k}$, $\forall n > n_0$.

Labeled Public-Key Encryption Schemes. A labeled public-key encryption scheme $\Pi = \{\mathsf{PPGen}^\Pi, \mathsf{KGen}^\Pi, \mathsf{Enc}^\Pi, \mathsf{Dec}^\Pi\}$ allows to encrypt a message m using a given public key pk_Π and label $\vartheta \in \{0,1\}^*$. In a nutshell, the given ciphertext leaks no information about the contained message, except its length, if the corresponding secret key sk_Π is not known. We require IND-CCA2 security to make our construction secure.

Sanitizable Signature Schemes. Subsequently, we restate the definitions of SSSs [3, 9, 34]. In a nutshell, a SSS allows a semi-trusted third party, named the sanitizer, to alter signer-chosen blocks to arbitrary bit-strings. The sanitizer holds its own key-pair and can be held accountable, if it sanitizes a message.

The following framework is essentially the one given by Camenisch et al. [14], which is itself based on existing work [9]. However, some additional notation is required beforehand. The variable $\mathsf{ADM}^{\mathsf{SSS}}$ contains the set of indices of the modifiable blocks, as well as ℓ, denoting the total number of blocks in the message m. For example, let $\mathsf{ADM}^{\mathsf{SSS}} = (\{1,2,4\}, 4)$. Then, m must contain four blocks ($\ell = 4$) and all but the third are

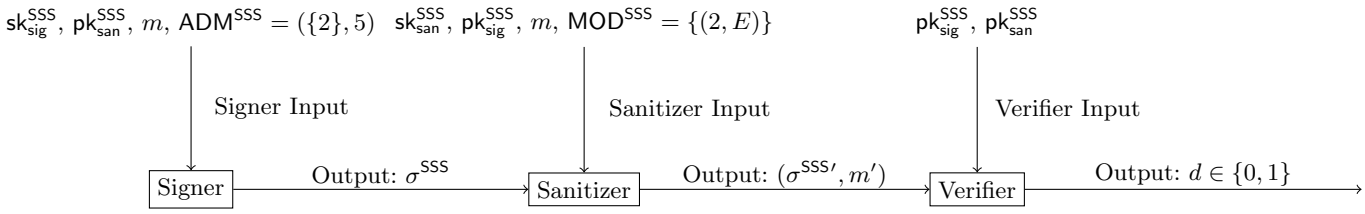


Fig. 1: Example workflow of an SSS. The message m is set to (H, A, L, L, O) and is sanitized to (H, E, L, L, O).

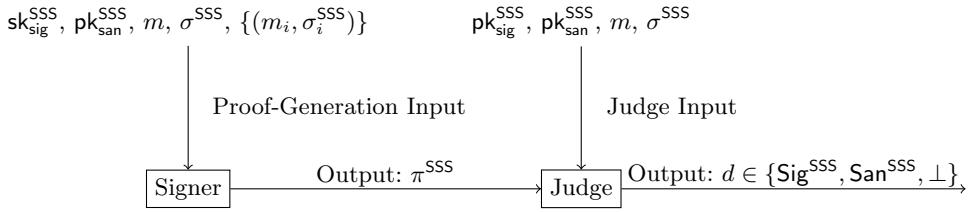


Fig. 2: Proof-generation and $\mathsf{Judge}^{\mathsf{SSS}}$

admissible. The variable $\mathsf{MOD}^{\mathsf{SSS}}$ is a set containing pairs $(i, m^{i\prime})$ for those blocks that are modified, meaning that m^i is replaced by $m^{i\prime}$. We use the shorthand notation $m' \leftarrow \mathsf{MOD}^{\mathsf{SSS}}(m)$ to denote the result of this replacement, while $\mathsf{MOD}^{\mathsf{SSS}} \prec (m, \mathsf{ADM}^{\mathsf{SSS}})$ means that $\mathsf{MOD}^{\mathsf{SSS}}$ is a valid modification instruction w.r.t. m and $\mathsf{ADM}^{\mathsf{SSS}}$. Likewise, we use $\mathsf{ADM}^{\mathsf{SSS}} \prec m$ to denote that $\mathsf{ADM}^{\mathsf{SSS}}$ is valid description of the admissible blocks w.r.t. m. An example workflow is depicted in Fig. 1 and Fig. 2. Both are derived from Bilzhause et al. [5].

Definition 1 (Sanitizable Signatures). *A sanitizable signature scheme* SSS *consists of the following eight ppt algorithms* $\{\mathsf{PPGen}^{\mathsf{SSS}}, \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{Sign}^{\mathsf{SSS}}, \mathsf{Verify}^{\mathsf{SSS}}, \mathsf{Sanitize}^{\mathsf{SSS}}, \mathsf{Proof}^{\mathsf{SSS}}, \mathsf{Judge}^{\mathsf{SSS}}\}$ *such that:*

$\mathsf{PPGen}^{\mathsf{SSS}}$. *The algorithm* $\mathsf{PPGen}^{\mathsf{SSS}}$ *generates the public parameters:*

$$\mathsf{pp}_{\mathsf{SSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{SSS}}(1^\kappa)$$

We assume that $\mathsf{pp}_{\mathsf{SSS}}$ *is implicitly input to all other algorithms.*

$\mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}$. *The algorithm* $\mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}$ *generates the key pair of the signer:*

$$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{SSS}})$$

$\mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}$. *The algorithm* $\mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}$ *generates the key pair of the sanitizer:*

$$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{SSS}})$$

$\mathsf{Sign}^{\mathsf{SSS}}$. *The algorithm* $\mathsf{Sign}^{\mathsf{SSS}}$ *generates a signature* σ^{SSS} *on input of the public key* $\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}$, $\mathsf{ADM}^{\mathsf{SSS}}$, *a message* m *and* $\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}$*:*

$$\sigma^{\mathsf{SSS}} \leftarrow_r \mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m, \mathsf{ADM}^{\mathsf{SSS}})$$

$\mathsf{Verify}^{\mathsf{SSS}}$. *The deterministic algorithm* $\mathsf{Verify}^{\mathsf{SSS}}$ *verifies a signature* σ^{SSS}*, i.e., outputs a decision* $d \in \{0, 1\}$ *w.r.t.* $\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}$, $\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}$ *and a message* m*:*

$$d \leftarrow \mathsf{Verify}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m, \sigma^{\mathsf{SSS}})$$

$\mathsf{Sanitize}^{\mathsf{SSS}}$. *The algorithm* $\mathsf{Sanitize}^{\mathsf{SSS}}$ *generates a sanitized signature* $\sigma^{\mathsf{SSS}\prime}$ *on input* $\mathsf{sk}_{\mathsf{san}}^{\mathsf{SSS}}$, $\mathsf{ADM}^{\mathsf{SSS}}$, *a message* m *and* $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{SSS}}$*:*

$$(m', \sigma^{\mathsf{SSS}\prime}) \leftarrow_r \mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{sk}_{\mathsf{san}}^{\mathsf{SSS}}, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{SSS}}, m, \sigma^{\mathsf{SSS}}, \mathsf{MOD}^{\mathsf{SSS}})$$

$\mathsf{Proof}^{\mathsf{SSS}}$. *The algorithm* $\mathsf{Proof}^{\mathsf{SSS}}$ *outputs a proof* π^{SSS} *on input* m, σ^{SSS}, $\mathsf{sk}_{\mathsf{sig}}^{\mathsf{SSS}}$, $\mathsf{pk}_{\mathsf{san}}^{\mathsf{SSS}}$ *and a set of polynomially many additional signature/message pairs* $\{(\sigma_i^{\mathsf{SSS}}, m_i)\}$*. The proof* π^{SSS} *is used by the next algorithm to pinpoint the accountable party for a given signature:*

$$\pi^{\mathsf{SSS}} \leftarrow_r \mathsf{Proof}^{\mathsf{SSS}}(\mathsf{sk}_{\mathsf{sig}}^{\mathsf{SSS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{SSS}}, m, \sigma^{\mathsf{SSS}}, \{(\sigma_i^{\mathsf{SSS}}, m_i)\})$$

$\mathsf{Judge}^{\mathsf{SSS}}$. *The deterministic algorithm* $\mathsf{Judge}^{\mathsf{SSS}}$ *outputs a decision* $d \in \{\mathsf{Sig}^{\mathsf{PS}}, \mathsf{San}^{\mathsf{PS}}, \perp\}$ *indicating whether the message/signature pair has been created by the signer, or the sanitizer:*

$$d \leftarrow \mathsf{Judge}^{\mathsf{SSS}}(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{SSS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{SSS}}, m, \sigma^{\mathsf{SSS}}, \pi^{\mathsf{SSS}})$$

Security Requirements. We only sketch the security requirements here for brevity. The formal game-based definitions are given in App. B. We stress that we use the strong definitions by Beck et al. [3].

Unforgeability. An outsider must not be able to create any new valid signatures.

Immutability. The sanitizer should, even if it can create it's own key pair, only sanitize admissible blocks and neither append or remove a block.

Privacy. An outsider not holding any secret keys should not be able to derive which message was contained before a sanitization took place.

Transparency. An outsider should not be able to decide whether a signature was created by the signer or the sanitizer.

Sanitizer-Accountability. The sanitizer should not be able to create a signature which points to the signer, even though the signer did not create it.

Signer-Accountability. The signer should not be able to create a proof for a signature which points to the sanitizer, even though the sanitizer did not create that signature.

Invisibility. An outsider not holding any secret keys should not be able to decide which parts of a signed message are admissible. It depends on the context if this notion is required.

Redactable Signature Schemes. The following definitions for RSSs are taken from Derler et al. [21], but extended to support parameter generation to match the definitions of SSSs. In particular, let $m = (m^1, m^2, \ldots, m^\ell)$ be some message, while $\mathsf{ADM}^{\mathsf{RSS}} \in \{1, 2, \ldots, \ell\}$ denotes the admissible redactions, i.e., if $i \in \mathsf{ADM}^{\mathsf{RSS}}$, then m^i can be redacted by *anyone*, i.e., no additional secrets are involved.[4] The variable $\mathsf{MOD}^{\mathsf{RSS}} \subseteq \{1, 2, \ldots, \ell\}$ denotes how a message m is to be modified, i.e., each block m^i, $i \in \mathsf{MOD}^{\mathsf{RSS}}$, is removed from m to form the redacted message m'. In comparison to Derler et al. [21], however, we already define how those data-structures look like for preciseness. Moreover, as done for SSSs, we use the shorthand notation $m' \leftarrow \mathsf{MOD}^{\mathsf{RSS}}(m)$ to denote a redaction. The notation $\mathsf{MOD}^{\mathsf{RSS}} \prec (m, \mathsf{ADM}^{\mathsf{RSS}})$ means that $\mathsf{MOD}^{\mathsf{RSS}}$ is a valid modification instruction w.r.t. m and $\mathsf{ADM}^{\mathsf{RSS}}$. Likewise, we use $\mathsf{ADM}^{\mathsf{RSS}} \prec m$ to denote that $\mathsf{ADM}^{\mathsf{RSS}}$ is valid description of the admissible blocks w.r.t. m. The "redaction information" $\mathsf{RED}^{\mathsf{RSS}}$ is some auxiliary string which may be used by some RSSs to improve the efficiency of the scheme [21]. We do not define what this string is, as we use RSSs as a black-box and thus it does not matter in our case.

An example workflow is depicted in Fig. 3, also taken from [5].

[4] Strictly speaking, there are approaches with additional secret keys, but they are not required for our construction [38].

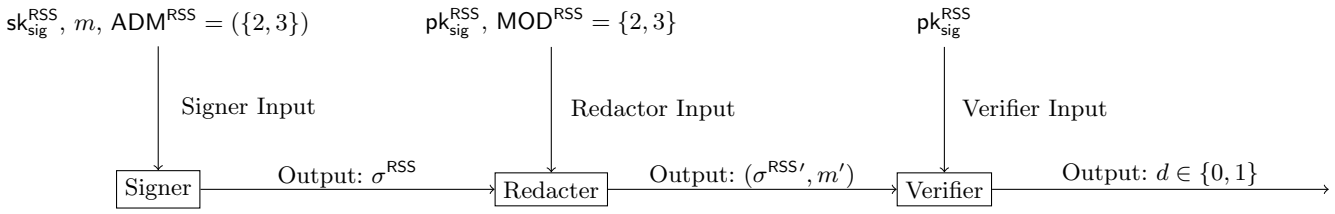


Fig. 3: Example workflow of an RSS. The message m is set to (I, do, not, like, crypto). After redacting, m' is (I, like, crypto). The redaction information $\mathsf{RED}^{\mathsf{RSS}}$ is omitted for brevity. As we only consider private RSSs, the redacted parts are not visible.

Definition 2 (Redactable Signatures). *A redactable signature scheme* RSS *consists of the following five algorithms, i.e.,* $\{\mathsf{PPGen}^{\mathsf{RSS}}, \mathsf{KGen}^{\mathsf{RSS}}, \mathsf{Sign}^{\mathsf{RSS}}, \mathsf{Verify}^{\mathsf{RSS}}, \mathsf{Redact}^{\mathsf{RSS}}\}$*, such that:*

$\mathsf{PPGen}^{\mathsf{RSS}}$**.** *The algorithm* $\mathsf{PPGen}^{\mathsf{RSS}}$ *generates the public parameters:*

$$\mathsf{pp}_{\mathsf{RSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{RSS}}(1^\kappa)$$

We assume that $\mathsf{pp}_{\mathsf{RSS}}$ *is implicitly input to all other algorithms.*

$\mathsf{KGen}^{\mathsf{RSS}}$**.** *The algorithm* $\mathsf{KGen}^{\mathsf{RSS}}$ *generates a key pair:*

$$(\mathsf{sk}_{\mathsf{sig}}^{\mathsf{RSS}}, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}}) \leftarrow_r \mathsf{KGen}^{\mathsf{RSS}}(\mathsf{pp}_{\mathsf{RSS}})$$

$\mathsf{Sign}^{\mathsf{RSS}}$**.** *The algorithm* $\mathsf{Sign}^{\mathsf{RSS}}$ *outputs a signature* σ^{RSS} *and some redaction information* $\mathsf{RED}^{\mathsf{RSS}}$ *on input of* $\mathsf{sk}_{\mathsf{sig}}^{\mathsf{RSS}}$, $\mathsf{ADM}^{\mathsf{RSS}}$ *and a message* m*:*

$$(\sigma^{\mathsf{RSS}}, \mathsf{RED}^{\mathsf{RSS}}) \leftarrow_r \mathsf{Sign}^{\mathsf{RSS}}(\mathsf{sk}_{\mathsf{sig}}^{\mathsf{RSS}}, m, \mathsf{ADM}^{\mathsf{RSS}})$$

Note, it is assumed that $\mathsf{ADM}^{\mathsf{RSS}}$ *can always be derived.*

$\mathsf{Verify}^{\mathsf{RSS}}$**.** *The deterministic algorithm* $\mathsf{Verify}^{\mathsf{RSS}}$ *verifies a signature* σ^{RSS}*, i.e., outputs a decision* $d \in \{0,1\}$ *w.r.t.* $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}}$ *and a message* m*:*

$$d \leftarrow \mathsf{Verify}^{\mathsf{RSS}}(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}}, m, \sigma^{\mathsf{RSS}})$$

$\mathsf{Redact}^{\mathsf{RSS}}$**.** *The algorithm* $\mathsf{Redact}^{\mathsf{RSS}}$ *outputs a derived signature* $\sigma^{\mathsf{RSS}\prime}$ *and a derived message* m' *on input of* $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}}$*, a signature* σ^{RSS}*, some modification instruction* $\mathsf{MOD}^{\mathsf{RSS}}$ *and some redaction information* $\mathsf{RED}^{\mathsf{RSS}}$*:*

$$(\sigma^{\mathsf{RSS}\prime}, m', \mathsf{RED}^{\mathsf{RSS}\prime}) \leftarrow_r \mathsf{Redact}^{\mathsf{RSS}}(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}}, m, \sigma^{\mathsf{RSS}}, \mathsf{MOD}^{\mathsf{RSS}}, \mathsf{RED}^{\mathsf{RSS}})$$

Note, to make our construction provably secure, we require that the redaction information $\mathsf{RED}^{\mathsf{RSS}}$ and its derivatives $\mathsf{RED}^{\mathsf{RSS}\prime}$, are always of constant size w.r.t. to κ. This is actually the case in the constructions by Derler et al. [21]. Note, Derler et al. require that even without $\mathsf{RED}^{\mathsf{RSS}}$ a redactor can redact [21]. Thus, this auxiliary string is only meant to make redactions more efficient.

Security Requirements. We only sketch the security requirements here for brevity. The formal game-based definitions are given in App. B.

Note, even though these definitions seem to be related to the ones for SSSs, one needs to take care that in standard definition of RSSs, only one key pair exists. Thus, neither accountability nor immutability are meaningful. Moreover, the unforgeability definition does not take signatures into account, but only messages, as everyone can redact.

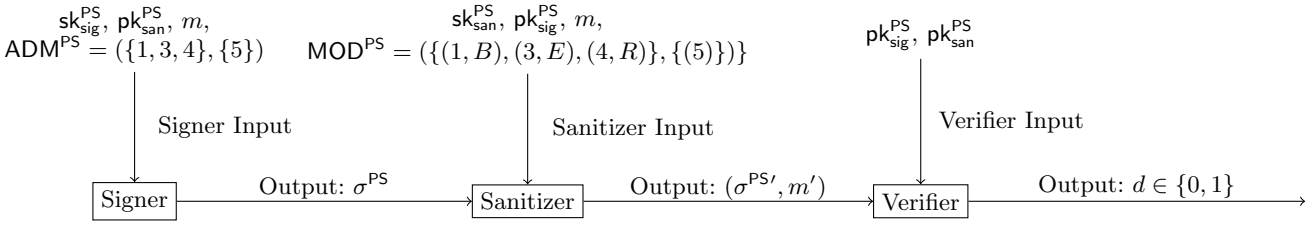


Fig. 4: Example workflow of a PS. The message m is set to (H, E, L, L, O) and is modified to (B, E, E, R).

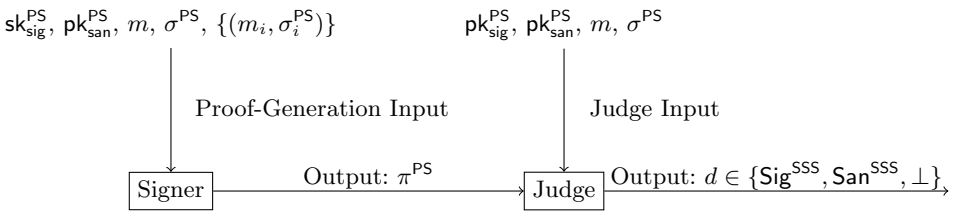


Fig. 5: Proof-generation and $\mathsf{Judge}^{\mathsf{PS}}$

Unforgeability. No one should, without holding the signing key, create valid signatures on messages not endorsed by the signer, i.e., forgeries exclude valid redactions.

Privacy. An outsider not holding any secret keys should not be able to derive which message was contained before a redaction took place.

Transparency. An outsider should not be able to decide whether a signature was the result of a redaction or not.

3 Protean Signatures

We now present our framework for PSs. To recap, a PS allows to remove and alter signer-chosen parts of a signed message by a semi-trusted third party, i.e., the sanitizer. The sanitizer can also be held accountable, if it chose to edit a signed message. For the framework, we need to settle some additional notation, which is derived from the ones used for RSSs and SSSs to ease understanding.

Protean Signature Schemes. For the framework, we use the following notation. The variable $\mathsf{ADM}^{\mathsf{PS}}$ is a list containing the set of indices of the editable blocks, as well as the blocks which can be redacted. For example, let $\mathsf{ADM}^{\mathsf{PS}} = (\{1,2\},\{4\})$. Then, the first and second block are editable, while only the fourth block can be redacted. The variable $\mathsf{MOD}^{\mathsf{PS}}$ is a list containing a set of pairs $(i, m^{i\prime})$ for those blocks that are modified, meaning that m^i is replaced by $m^{i\prime}$ and a set of indices to be redacted. In more detail, if $\mathsf{MOD}^{\mathsf{PS}} = (\{(1,b),(2,b)\},\{3\})$ means that the first two blocks are altered to contain a b, while the third block is redacted.

We use the shorthand notation $m' \leftarrow \mathsf{MOD}^{\mathsf{PS}}(m)$ to denote the result of this replacement, while $\mathsf{MOD}^{\mathsf{PS}} \prec (m, \mathsf{ADM}^{\mathsf{PS}})$ means that $\mathsf{MOD}^{\mathsf{PS}}$ is a valid modification instruction w.r.t. m and $\mathsf{ADM}^{\mathsf{PS}}$. Likewise, we use $\mathsf{ADM}^{\mathsf{PS}} \prec m$ to denote that $\mathsf{ADM}^{\mathsf{PS}}$ is valid description of the admissible blocks w.r.t. m.

An example workflow is depicted in Fig. 4 and Fig. 5. Note, that this is very similar to SSSs. To ease understanding and the description of our construction, we define that the replacements are done first and the redactions afterwards.

Definition 3 (Protean Signature). *A Protean Signature scheme* PS *consists of the following eight ppt algorithms* $(\mathsf{PPGen}^{\mathsf{PS}}, \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{Sign}^{\mathsf{PS}}, \mathsf{Verify}^{\mathsf{PS}}, \mathsf{Edit}^{\mathsf{PS}}, \mathsf{Proof}^{\mathsf{PS}}, \mathsf{Judge}^{\mathsf{PS}})$ *such that:*

$\mathsf{PPGen}^{\mathsf{PS}}$**.** *The algorithm* $\mathsf{PPGen}^{\mathsf{PS}}$ *generates the public parameters:*

$$\mathsf{pp}_{\mathsf{PS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{PS}}(1^\kappa)$$

We assume that $\mathsf{pp}_{\mathsf{PS}}$ *is implicitly input to all other algorithms.*

$\mathsf{KGen}^{\mathsf{PS}}_{\mathsf{sig}}$**.** *The algorithm* $\mathsf{KGen}^{\mathsf{PS}}_{\mathsf{sig}}$ *generates the key pair of the signer:*

$$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{PS}})$$

$\mathsf{KGen}^{\mathsf{PS}}_{\mathsf{san}}$**.** *The algorithm* $\mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}$ *generates the key pair of the sanitizer:*

$$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{PS}})$$

$\mathsf{Sign}^{\mathsf{PS}}$**.** *The algorithm* $\mathsf{Sign}^{\mathsf{PS}}$ *generates a signature* σ^{PS} *on input of the public key* $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}$, $\mathsf{ADM}^{\mathsf{PS}}$, *a message* m, *and* $\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}$*:*

$$\sigma^{\mathsf{PS}} \leftarrow_r \mathsf{Sign}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \mathsf{ADM}^{\mathsf{PS}})$$

It is assumed that $\mathsf{ADM}^{\mathsf{PS}}$ *can be derived from any verifying signature* σ^{PS}*, if* $\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}$ *is known.*

$\mathsf{Verify}^{\mathsf{PS}}$**.** *The deterministic algorithm* $\mathsf{Verify}^{\mathsf{PS}}$ *verifies a signature* σ^{PS}*, i.e., outputs a decision* $d \in \{0,1\}$ *w.r.t.* $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}$, $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}$, *and a message* m*:*

$$d \leftarrow \mathsf{Verify}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \sigma^{\mathsf{PS}})$$

$\mathsf{Edit}^{\mathsf{PS}}$**.** *The algorithm* $\mathsf{Edit}^{\mathsf{PS}}$ *generates a sanitized signature* $\sigma^{\mathsf{PS}\prime}$ *and updated* $\mathsf{ADM}^{\mathsf{PS}\prime}$*, given inputs* $\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}$, $\mathsf{ADM}^{\mathsf{PS}}$, *a message* m, *and* $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}$*:*

$$(m', \sigma^{\mathsf{PS}\prime}, \mathsf{ADM}^{\mathsf{PS}\prime}) \leftarrow_r \mathsf{Edit}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, m, \mathsf{MOD}^{\mathsf{PS}})$$

$\mathsf{Proof}^{\mathsf{PS}}$**.** *The algorithm* $\mathsf{Proof}^{\mathsf{PS}}$ *outputs a proof* π^{PS} *on input* m, σ^{PS}, $\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}$, $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}$, *and a set of polynomially many additional signature/message pairs* $\{(\sigma^{\mathsf{PS}}_i, m^i)\}$*. The proof* π^{PS} *is used by the next algorithm to pinpoint the accountable party for a given signature:*

$$\pi^{\mathsf{PS}} \leftarrow_r \mathsf{Proof}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \sigma^{\mathsf{PS}}, \{(\sigma^{\mathsf{PS}}_i, m^i)\})$$

$\mathsf{Judge}^{\mathsf{PS}}$**.** *The deterministic algorithm* $\mathsf{Judge}^{\mathsf{PS}}$ *outputs a decision* $d \in \{\mathsf{Sig}^{\mathsf{PS}}, \mathsf{San}^{\mathsf{PS}}, \perp\}$ *indicating whether the message/signature pair has been created by the signer, or the sanitizer:*

$$d \leftarrow \mathsf{Judge}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \sigma^{\mathsf{PS}}, \pi^{\mathsf{PS}})$$

PSs Security Definitions. We now introduce the security properties for PSs. Clearly, the goals are similar to the ones for SSSs and RSSs. However, due to the extended capabilities, the semantic is quite different, while we need to take extra care for changed indices after redactions.

Unforgeability. This definition requires that an adversary $\mathcal{A}$ not having any secret keys is not able to produce any valid signature $\sigma^{\mathsf{PS}*}$ on a message m^* which it has never not seen, even if $\mathcal{A}$ has full oracle access, i.e., this captures "strong unforgeability" [34].

Definition 4 (Unforgeability). *A* PS *is unforgeable, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{Unforgeability}^{\mathsf{PS}}_{\mathcal{A}}(\kappa) = 1] \leq \nu(\kappa)$*,where the corresponding experiment is defined in Fig. 6.*

Experiment $\mathsf{Unforgeability}^{\mathsf{PS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{PS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{PS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{PS}})$
$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{PS}})$
$(m^*, \sigma^{\mathsf{PS}*}) \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Edit}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}},\cdot,\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$
for $i = 1, 2, \ldots, q$ let $(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san},i}, m_i, \mathsf{ADM}^{\mathsf{PS}}_i)$ and σ^{PS}_i
index the queries/answers to/from $\mathsf{Sign}^{\mathsf{PS}}$
for $j = 1, 2, \ldots, q'$ let $(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig},j}, m_j, \sigma^{\mathsf{PS}}_j, \mathsf{MOD}_j)$ and $(m'_j, \sigma^{\mathsf{PS}\prime}_j, \mathsf{ADM}^{\mathsf{PS}\prime}_j)$
index the queries/answers to/from $\mathsf{Edit}^{\mathsf{PS}}$
return 1, if $\mathsf{Verify}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m^*, \sigma^{\mathsf{PS}*}) = 1 \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m^*, \sigma^{\mathsf{PS}*}) \neq (\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san},i}, m_i, \sigma^{\mathsf{PS}}_i) \wedge$
$\forall j \in \{1, 2, \ldots, q'\} : (\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, m^*, \sigma^{\mathsf{PS}*}) \neq (\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig},j}, m'_j, \sigma^{\mathsf{PS}\prime}_j)$
return 0

Fig. 6: PS Unforgeability

Experiment $\mathsf{Immutability}^{\mathsf{PS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{PS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{PS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{PS}})$
$(m^*, \sigma^{\mathsf{PS}*}, \mathsf{pk}^{\mathsf{PS}*}_{\mathsf{san}}) \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}})$
for $i = 1, 2, \ldots, q$ let $(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san},i}, m_i, \mathsf{ADM}^{\mathsf{PS}}_i)$
index the queries to $\mathsf{Sign}^{\mathsf{PS}}$
return 1, if $\mathsf{Verify}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}*}_{\mathsf{san}}, m^*, \sigma^{\mathsf{PS}*}) = 1 \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^{\mathsf{PS}*}_{\mathsf{san}} \neq \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san},i} \vee$
$m^* \notin \{\mathsf{MOD}(m_i) \mid \mathsf{MOD} \text{ with } \mathsf{MOD} \prec (m_i, \mathsf{ADM}^{\mathsf{PS}}_i)\})$
return 0

Fig. 7: PS Immutability

Immutability. This definition prohibits that an adversary $\mathcal{A}$ can generate a verifying signature $\sigma^{\mathsf{PS}*}$ for a message m^* not derivable from the signatures given by an honest signer, even if it can generate the editor's key pair.

Definition 5 (Immutability). *A* PS *is immutable, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Immutability}^{\mathsf{PS}}_{\mathcal{A}}(\kappa) = 1] \leq \nu(\kappa)$ *,where the corresponding experiment is defined in Fig. 7.*

Privacy. This definition prohibits that an adversary $\mathcal{A}$ can learn anything about edited (redacted or sanitized) parts.

Definition 6 (Privacy). *A* PS *is private, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that* $\left|\Pr[\mathsf{Privacy}^{\mathsf{PS}}_{\mathcal{A}}(\kappa)] - 1/2\right| \leq \nu(\kappa)$ *,where the corresponding experiment is defined in Fig. 8.*

Transparency. This definition requires that an adversary $\mathcal{A}$ does not learn whether a signature σ^{PS} was generated through $\mathsf{Sign}^{\mathsf{PS}}$ or $\mathsf{Edit}^{\mathsf{PS}}$.

Definition 7 (Transparency). *A* PS *is transparent, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that* $\left|\Pr[\mathsf{Transparency}^{\mathsf{PS}}_{\mathcal{A}}(\kappa)] - 1/2\right| \leq \nu(\kappa)$ *,where the corresponding experiment is defined in Fig. 9.*

Experiment $\mathsf{Privacy}^{\mathsf{PS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{PS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{PS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{PS}})$
$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{PS}})$
$b \leftarrow_r \{0,1\}$
$a \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Edit}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}},\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{LoREdit}(\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}},b)}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$
where oracle $\mathsf{LoREdit}$ on input of $m_0, m_1, \mathsf{MOD}^{\mathsf{PS}}_0, \mathsf{MOD}^{\mathsf{PS}}_1, \mathsf{ADM}^{\mathsf{PS}}_0, \mathsf{ADM}^{\mathsf{PS}}_1, \mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, b$
let $\sigma^{\mathsf{PS}}_i \leftarrow_r \mathsf{Sign}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m_i, \mathsf{ADM}^{\mathsf{PS}}_i)$ for $i \in \{0,1\}$
let $(m'_i, \sigma^{\mathsf{PS}\prime}_i, \mathsf{ADM}^{\mathsf{PS}\prime}_i) \leftarrow_r \mathsf{Edit}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, m_i, \sigma^{\mathsf{PS}}_i, \mathsf{MOD}^{\mathsf{PS}}_i)$ for $i \in \{0,1\}$
return $\bot$, if $m'_0 \neq m'_1 \ \vee \ \mathsf{ADM}^{\mathsf{PS}\prime}_0 \neq \mathsf{ADM}^{\mathsf{PS}\prime}_1$
return $(m'_b, \sigma^{\mathsf{PS}\prime}_b, \mathsf{ADM}^{\mathsf{PS}\prime}_b)$
return 1, if $a = b$
return 0

Fig. 8: PS Privacy

Experiment $\mathsf{Transparency}^{\mathsf{PS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{PS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{PS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{PS}})$
$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{PS}})$
$b \leftarrow_r \{0,1\}$
$\mathcal{Q} \leftarrow \emptyset$
$a \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Edit}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}},\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{PS}\prime}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{Sign/Edit}(\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}},b)}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$
where oracle $\mathsf{Proof}^{\mathsf{PS}\prime}$ on input of $\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, m, \sigma^{\mathsf{PS}}, \{(m_i, \sigma^{\mathsf{PS}}_i) \mid i \in \mathbb{N}\}$:
return $\bot$, if $\mathsf{pk}^{\mathsf{PS}\prime}_{\mathsf{san}} = \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}} \ \wedge \ ((m, \sigma^{\mathsf{PS}}) \in \mathcal{Q} \ \vee \ \mathcal{Q} \cap \{(m_i, \sigma^{\mathsf{PS}}_i)\} \neq \emptyset)$
return $\mathsf{Proof}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}\prime}_{\mathsf{san}}, m, \sigma^{\mathsf{PS}}, \{(m_i, \sigma^{\mathsf{PS}}_i)\})$
where oracle $\mathsf{Sign/Edit}$ on input of $m, \mathsf{MOD}^{\mathsf{PS}}, \mathsf{ADM}^{\mathsf{PS}}, \mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, b$:
$\sigma^{\mathsf{PS}} \leftarrow_r \mathsf{Sign}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \mathsf{ADM}^{\mathsf{PS}})$
$(m', \sigma^{\mathsf{PS}\prime}, \mathsf{ADM}^{\mathsf{PS}\prime}) \leftarrow_r \mathsf{Edit}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, m, \sigma^{\mathsf{PS}}, \mathsf{MOD}^{\mathsf{PS}})$
if $b = 1$:
$\sigma^{\mathsf{PS}\prime} \leftarrow_r \mathsf{Sign}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m', \mathsf{ADM}^{\mathsf{PS}\prime})$
if $\sigma^{\mathsf{PS}\prime} \neq \bot$, set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m', \sigma^{\mathsf{PS}\prime})\}$
return $(m', \sigma^{\mathsf{PS}\prime})$
return 1, if $a = b$
return 0

Fig. 9: PS Transparency

Signer-Accountability. Signer-accountability prohibits that an adversary can generate a bogus proof that makes $\mathsf{Judge}^{\mathsf{PS}}$ decide that the sanitizer is responsible for a given signature/message pair $(m^*, \sigma^{\mathsf{PS}*})$, but the sanitizer has never generated this pair. This is even true, if the adversary can generate the signer's key pair.

Definition 8 (Signer-Accountability). *A* PS *is signer-accountable, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*
$\Pr[\mathsf{SigAccountability}^{\mathsf{PS}}_{\mathcal{A}}(\kappa) = 1] \leq \nu(\kappa)$ *,where the corresponding experiment is defined in Fig. 10.*

Sanitizer-Accountability. Sanitizer-accountability prohibits that an adversary can generate a bogus signature/message pair $(m^*, \sigma^{\mathsf{PS}*})$ that makes $\mathsf{Proof}^{\mathsf{SSS}}$ output an honestly generated proof π^{PS} which points to

Experiment $\mathsf{SigAccountability}^{\mathsf{PS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{PS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{PS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{PS}})$
$(\mathsf{pk}^{\mathsf{PS}*}_{\mathsf{sig}}, \pi^{\mathsf{PS}*}, m^*, \sigma^{\mathsf{PS}*}) \leftarrow_r \mathcal{A}^{\mathsf{Edit}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$
for $i = 1, 2, \ldots, q$ let $(m'_i, \sigma^{\mathsf{PS}\prime}_i, \mathsf{ADM}^{\mathsf{PS}\prime}_j)$ and $(m_i, \mathsf{MOD}^{\mathsf{PS}}_i, \sigma^{\mathsf{PS}}_i, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig},i})$
index the answers/queries from/to $\mathsf{Edit}^{\mathsf{PS}}$
return 1, if $\mathsf{Verify}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}*}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m^*, \sigma^{\mathsf{PS}*}) = 1 \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^{\mathsf{PS}*}_{\mathsf{sig}}, m^*, \sigma^{\mathsf{PS}*}) \neq (\mathsf{pk}_{\mathsf{sig}}{}^{\mathsf{PS}}{}_{,i}, m'_i, \sigma^{\mathsf{PS}\prime}_i) \wedge$
$\mathsf{Judge}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}*}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m^*, \sigma^{\mathsf{PS}*}, \pi^{\mathsf{PS}*}) = \mathsf{San}^{\mathsf{PS}}$
return 0

Fig. 10: PS Signer-Accountability

Experiment $\mathsf{SanAccountability}^{\mathsf{PS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{PS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{PS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{PS}})$
$(m^*, \sigma^{\mathsf{PS}*}, \mathsf{pk}^{\mathsf{PS}\,*}_{\mathsf{san}}) \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}})$
for $i = 1, 2, \ldots, q$ let $(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san},i}, m_i, \mathsf{ADM}^{\mathsf{PS}}_i)$ and σ^{PS}_i
index the queries/answers to/from $\mathsf{Sign}^{\mathsf{PS}}$
$\pi^{\mathsf{PS}} \leftarrow_r \mathsf{Proof}^{\mathsf{PS}}(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}\,*}_{\mathsf{san}}, m^*, \sigma^{\mathsf{PS}*}, \{(m_i, \sigma^{\mathsf{PS}}_i) \mid 0 < i \leq q\})$
return 1, if $\mathsf{Verify}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}\,*}_{\mathsf{san}}, m^*, \sigma^{\mathsf{PS}*}) = 1 \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^{\mathsf{PS}\,*}_{\mathsf{san}}, m^*, \sigma^{\mathsf{PS}*}) \neq (\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san},i}, m_i, \sigma^{\mathsf{PS}}_i) \wedge$
$\mathsf{Judge}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}\,*}_{\mathsf{san}}, m^*, \sigma^{\mathsf{PS}*}, \pi^{\mathsf{PS}}) = \mathsf{Sig}^{\mathsf{PS}}$
return 0

Fig. 11: PS Sanitizer-Accountability

the signer, but $(m^*, \sigma^{\mathsf{PS}*})$ has never been generated by the signer. This is even true, if the adversary can generate the sanitizer's key pair.

Definition 9 (Sanitizer-Accountability). *A* PS *is sanitizer-accountable, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{SanAccountability}^{\mathsf{PS}}_{\mathcal{A}}(\kappa) = 1] \leq \nu(\kappa)$ *,where the corresponding experiment is defined in Fig. 11.*

Invisibility. Invisibility prohibits that an outsider can decide which blocks can be edited. However, as there are no RSSs which prohibit to decide which elements are redactable (as redactions are always public), we restrict ourselves to the case of which blocks can be edited by the sanitizer. This is a very strong privacy notion and might be required, or not, depending on the use-case. However, we want to point out that achieving this notion is possible with our construction, even though it comes at a high price (cf. Sect. 4). The main idea is that the adversary has to decide which blocks are admissible — the oracle either uses $\mathsf{ADM}^{\mathsf{PS}}_0.1$ or $\mathsf{ADM}^{\mathsf{PS}}_1.1$. Here, $\mathsf{ADM}^{\mathsf{PS}}_1.b$, where $b \in \{1, 2\}$ means the bth element of the list. To avoid trivial attacks, the adversary needs to be limited to further edit messages where $(\mathsf{ADM}^{\mathsf{PS}}_0.1 \cap \mathsf{ADM}^{\mathsf{PS}}_1.1)$. Note, the signing oracle can be simulated by using the same $\mathsf{ADM}^{\mathsf{PS}}$ in the LoRADM oracle.

Definition 10 (Invisibility). *A* PS *is invisible, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\left|\Pr[\mathsf{Invisibility}^{\mathsf{PS}}_{\mathcal{A}}(\kappa)] - 1/2\right| \leq \nu(\kappa)$ *,where the corresponding experiment is defined in Fig. 12.*

We conclude with a final definition.

Experiment $\mathsf{Invisibility}^{\mathsf{PS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{PS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{PS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{PS}})$
$(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{PS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{PS}})$
$b \leftarrow_r \{0,1\}$
$\mathcal{Q} \leftarrow \emptyset$
$a \leftarrow_r \mathcal{A}^{\mathsf{Edit}^{\mathsf{PS}\prime}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}},\cdot,\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{LoRADM}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot,b)}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$
where oracle LoRADM on input of $\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}\prime}_{\mathsf{san}}, m, \mathsf{ADM}^{\mathsf{PS}}_0, \mathsf{ADM}^{\mathsf{PS}}_1, b$:
return $\bot$, if $\mathsf{ADM}^{\mathsf{PS}}_0.2 \neq \mathsf{ADM}^{\mathsf{PS}}_1.2 \ \vee \ \mathsf{ADM}^{\mathsf{PS}}_0 \not\preceq m \wedge \mathsf{ADM}^{\mathsf{PS}}_1 \not\preceq m$
return $\bot$, if $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}} \neq \mathsf{pk}^{\mathsf{PS}\prime}_{\mathsf{san}} \ \wedge \ \mathsf{ADM}^{\mathsf{PS}}_0 \neq \mathsf{ADM}^{\mathsf{PS}}_1$
let $\sigma^{\mathsf{PS}} \leftarrow_r \mathsf{Sign}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}\prime}_{\mathsf{san}}, m, \mathsf{ADM}^{\mathsf{PS}}_b)$
if $\mathsf{pk}^{\mathsf{PS}\prime}_{\mathsf{san}} = \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}$, let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m, \sigma^{\mathsf{PS}}, ((\mathsf{ADM}^{\mathsf{PS}}_0.1 \cap \mathsf{ADM}^{\mathsf{PS}}_1.1), \mathsf{ADM}^{\mathsf{PS}}_0.2))\}$
return σ^{PS}
where oracle $\mathsf{Edit}^{\mathsf{PS}\prime}$ on input of $\mathsf{pk}^{\mathsf{PS}\prime}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, m, \mathsf{MOD}^{\mathsf{PS}}, \sigma^{\mathsf{PS}}$:
return $\bot$, if $\mathsf{pk}^{\mathsf{PS}\prime}_{\mathsf{sig}} = \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}} \wedge \nexists(m, \sigma^{\mathsf{PS}}, \mathsf{ADM}) \in \mathcal{Q} : \mathsf{MOD}^{\mathsf{PS}} \prec (m, \mathsf{ADM})$
let $(m', \sigma^{\mathsf{PS}\prime}, \mathsf{ADM}^{\mathsf{PS}\prime\prime}) \leftarrow_r \mathsf{Edit}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}\prime}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, m, \mathsf{MOD}^{\mathsf{PS}}, \sigma^{\mathsf{PS}})$
if $\mathsf{pk}^{\mathsf{PS}\prime}_{\mathsf{sig}} = \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}} \wedge \exists(m, \sigma^{\mathsf{PS}}, \mathsf{ADM}^{\mathsf{PS}\prime}) \in \mathcal{Q} : \mathsf{MOD}^{\mathsf{PS}} \prec (m, \mathsf{ADM}^{\mathsf{PS}\prime})$,
let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m', \sigma^{\mathsf{PS}\prime}, \mathsf{ADM}^{\mathsf{PS}\prime\prime})\}$
return $(m', \sigma^{\mathsf{PS}\prime})$
return 1, if $a = b$
return 0

Fig. 12: PS Invisibility

Definition 11 (Secure PSs). *A* PS *is secure, if it unforgeable, private, transparent, immutable, signer-accountable, and sanitizer-accountable.*

If invisibility is required, depends on the use-case.

Are PSs a Generalization? We now show that PSs are indeed a generalization of both SSSs and RSSs. In particular, it suffices to show how to construct SSSs and RSSs from a PS. Formalized, we have the following theorem:

Theorem 1. *Secure* SSS*s and* RSS*s can be black-box constructed from* PS*s.*

Proof. We prove this theorem by providing two black-box constructions which "transform" any secure PS into a SSS (or RSS resp.). Thus, we split the proof in two parts, each giving a concrete construction.

SSS*s from* PS. We first show how an SSS can be constructed from a PS. This is actually straightforward; an SSS can be emulated by prohibiting any redactions.

RSS*s from* PS. This follows from the fact that SSSs (which we now know how to build from PSs) can emulate standard signature schemes, i.e., by prohibiting any sanitizations by the sanitizer. Moreover, RSSs as defined above can be build from standard signature schemes [8]. Thus, the theorem follows.

We stress that Brzuska et al. [8] address trees, but the underlying idea can trivially be extended for lists as well. However, they do not allow that a signer can prohibit certain redactions. This can easily be solved by adding a special marked block (e.g., the first one) which contains all blocks which cannot be redacted. A black-box construction of such a RSS is given in App. C.

4 Construction

In this section, we introduce our construction for PSs. The basic idea is to combine RSSs and SSSs by bridging them using unique tags. In more detail, each block $m^i \in m$ is signed using an SSS, while an additional (non-admissible) tag τ is used to identify the "overall" message m the block m^i belongs to. Moreover, each block m^i is also assigned a (non-admissible) additional tag τ_i, along with all public keys, used by the RSS to allow for redactions. Thus, there are ℓ_m σ_i^{SSS}, where each signature protects $(m^i, \tau, \tau_i, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$. If a block m_i is sanitizable, it is marked as admissible within $\mathsf{ADM}_i^{\mathsf{SSS}}$. This allows to sanitize the block m^i. Then, each tag τ_i is put into an RSS to allow for transparent redactions, additionally bound to the non-redactable "overall" tag τ and all (non-redactable) public keys. If a block m^i is non-redactable, this is marked in $\mathsf{ADM}^{\mathsf{RSS}}$. Thus, σ^{RSS} protects $(\tau_1, \ldots, \tau_{\ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$. For technical reason, namely to make the auxiliary redaction information $\mathsf{RED}^{\mathsf{RSS}}$ available to the sanitizer, we also encrypt this information as c; to achieve stateless signers and sanitizers, this information is also "self-encrypted" by the redactor itself upon editing. Finally, to achieve accountability, all tags, all signatures generated so far, the resulting ciphertext and public keys are signed again using an additional SSS, while in this outer SSS everything, but the public keys and the tag τ are admissible. To maintain transparency, the overall message m is a single block in the outer SSS.

In more detail, the outer signature σ_0^{SSS} protects $(m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_i, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$. Thus, changing the message or any signature requires changing σ_0^{SSS}, implying accountability. Upon sanitization of a block m^i, σ_i^{SSS} is sanitized, while the outer signature σ_0^{SSS} needs to be adjusted as well. For redaction of a block m^i, σ^{RSS} is adjusted and the corresponding signature is no longer given out. This also means that σ_0^{SSS} must be adjusted.

Our resulting construction is depicted in Construction 1. To give a graphical overview of the construction idea, see Fig. 13 (before editing) and Fig. 14 (after editing). We stress that this idea seems to be straightforward, but there are a lot of details to make the construction provably secure. Moreover, we do not consider unlinkability [11], as it seems to be very hard to achieve with the underlying construction paradigm, especially considering that there are no SSSs yet which are unlinkable and invisible at the same time.

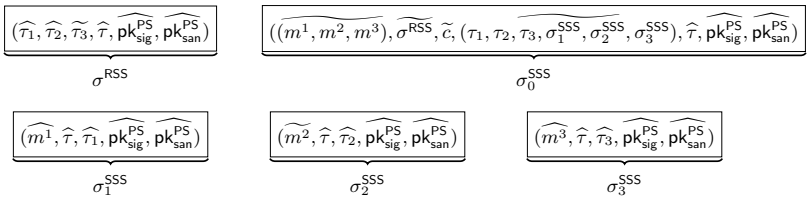


Fig. 13: Our main construction idea. Let $\mathsf{ADM}^{\mathsf{PS}} = (\{2\}, \{3\})$ and $m = (m^1, m^2, m^3)$ for preciseness, i.e., only the second block of m is sanitizable, while only the last block of m is redactable. Redactable elements for the RSS (or sanitizable for the SSS) are marked with a tilde, i.e., $\widetilde{\cdot}$. Blocks which are not redactable (or sanitizable resp.) are marked with a hat, i.e., $\widehat{\cdot}$.

Security. The proof of the following Theorem is found in Appendix E.

Theorem 2. *The scheme given in Construction 1 is secure (and invisible), if Π,* SSS *and* RSS *are secure (and* SSS *is also invisible).*

Proof (Sketch). Transparency and privacy follow from the transparency and privacy of the underlying primitives, the uniform distribution of the tags, and the IND-CCA2-security of Π. Immutability follows from the unforgeability of the RSS and the immutability of the SSSs. Accountability directly follows from the accountability of the outer SSS. Likewise, invisibility follows from the invisibility of the used SSSs.

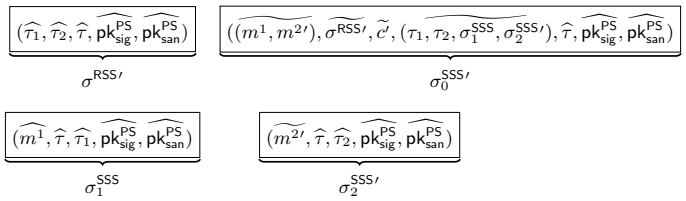


Fig. 14: State after sanitization. Here, block m^3 was redacted and m^2 was changed to $m^{2\prime}$. Block m^1 must stay the same.

$\mathsf{PPGen}^{\mathsf{PS}}(1^\kappa)$. Let $\mathsf{pp}_\Pi \leftarrow_r \mathsf{PPGen}^\Pi(1^\kappa)$, $\mathsf{pp}_{\mathsf{SSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{SSS}}(1^\kappa)$ and $\mathsf{pp}_{\mathsf{RSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{RSS}}(1^\kappa)$. Return $\mathsf{pp}_{\mathsf{PS}} = (\mathsf{pp}_\Pi, \mathsf{pp}_{\mathsf{SSS}}, \mathsf{pp}_{\mathsf{RSS}})$.

$\mathsf{KGen}^{\mathsf{PS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{PS}})$. Let $(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{SSS}})$ and $(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{Sign}^{\mathsf{RSS}}(\mathsf{pp}_{\mathsf{RSS}})$. Return $(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}) = ((\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}), (\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}))$.

$\mathsf{KGen}^{\mathsf{PS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{PS}})$. Let $(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{SSS}})$ and $(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) \leftarrow_r \mathsf{KGen}^\Pi(\mathsf{pp}_\Pi)$. Return $(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}) = ((\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{sk}_\Pi), (\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}_\Pi))$.

$\mathsf{Sign}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \mathsf{ADM}^{\mathsf{PS}})$. If $\mathsf{ADM}^{\mathsf{PS}} \prec m$, where $m = (m^1, m^2, \ldots, m^\ell)$, continue. Otherwise, return $\perp$. Parse $\mathsf{ADM}^{\mathsf{PS}} = (\mathsf{ADM}^{\mathsf{PS}}_1, \mathsf{ADM}^{\mathsf{PS}}_2)$. Draw $\tau \leftarrow_r \{0,1\}^\kappa$. Do $(\forall i \in \{1, 2, \ldots, \ell_m\})$: $\sigma_i^{\mathsf{SSS}} \leftarrow_r \mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, (m^i, \tau, \tau_i, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}), \mathsf{ADM}^{\mathsf{SSS}}_i)$, where each $\tau_i \leftarrow_r \{0,1\}^\kappa$ and, if $i \in \mathsf{ADM}^{\mathsf{PS}}_1$, let $\mathsf{ADM}^{\mathsf{SSS}}_i = (\{1\}, 5)$ and $\mathsf{ADM}^{\mathsf{SSS}}_i = (\emptyset, 5)$ otherwise. Next, let $(\sigma^{\mathsf{RSS}}, \mathsf{RED}^{\mathsf{RSS}}) \leftarrow_r \mathsf{Sign}^{\mathsf{RSS}}(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, m', \mathsf{ADM}^{\mathsf{RSS}})$, where $\mathsf{ADM}^{\mathsf{RSS}} = \mathsf{ADM}^{\mathsf{PS}}_2$ and the message $m' = (\tau_1, \tau_2, \ldots, \tau_\ell, \tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$. Next, let $c \leftarrow_r \mathsf{Enc}^\Pi(\mathsf{pk}_\Pi, \mathsf{RED}^{\mathsf{RSS}}, (\tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}))$. Finally, generate the signature $\sigma_0^{\mathsf{SSS}} \leftarrow_r \mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, (m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1 \le i \le \ell_m}, \tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}), \mathsf{ADM}^{\mathsf{SSS}}_0)$, where $\mathsf{ADM}^{\mathsf{SSS}}_0 = (\{1, 2, 3, 4\}, 7)$. Return $((\sigma_i^{\mathsf{SSS}})_{0 \le i \le \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1 \le i \le \ell_m})$.

$\mathsf{Verify}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \sigma^{\mathsf{PS}})$. If $\mathsf{Verify}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, (m, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1 \le i \le \ell_m}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}), \sigma_0^{\mathsf{SSS}}) = 0$, return 0. If $0 = \mathsf{Verify}^{\mathsf{RSS}}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}, (\tau_1, \tau_2, \ldots, \tau_\ell, \tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}), \sigma^{\mathsf{RSS}})$, return 0. If for any $i \in \{1, 2, \ldots, \ell_m\} : 0 = \mathsf{Verify}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, (m^i, \tau, \tau_i, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}), \sigma_i^{\mathsf{SSS}})$, return 0. Return 1.

$\mathsf{Edit}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, m, \mathsf{MOD}^{\mathsf{PS}})$. If $0 = \mathsf{Verify}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \sigma^{\mathsf{PS}})$, return $\perp$. Parse $\mathsf{MOD}^{\mathsf{PS}} = (\mathsf{MOD}^{\mathsf{PS}}_1, \mathsf{MOD}^{\mathsf{PS}}_2)$. Further do $(\forall (i, m^{i\prime}) \in \mathsf{MOD}^{\mathsf{PS}}_1)$: $(m^{i\prime}, \sigma_i^{\mathsf{SSS}\prime}) \leftarrow_r \mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, (m^i, \tau, \tau_i, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}), \sigma_i^{\mathsf{SSS}}, \{(0, m^{i\prime})\})$. If any $\sigma_i^{\mathsf{SSS}\prime} = \perp$, return $\perp$. Let $\vartheta = (\tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$ and $\mathsf{RED}^{\mathsf{RSS}} \leftarrow \mathsf{Dec}^\Pi(\mathsf{sk}_\Pi, c, \vartheta)$. If $\mathsf{RED}^{\mathsf{RSS}} = \perp$, return $\perp$. Generate $(\sigma^{\mathsf{RSS}\prime}, m', \mathsf{RED}^{\mathsf{RSS}\prime}) \leftarrow_r \mathsf{Redact}^{\mathsf{RSS}}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}, m'', \sigma^{\mathsf{RSS}}, \mathsf{MOD}^{\mathsf{PS}}_2, \mathsf{RED}^{\mathsf{RSS}})$, where $m'' = (\tau_1, \tau_2, \ldots, \tau_\ell, \tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$. If $\sigma^{\mathsf{RSS}\prime} = \perp$, return $\perp$. Let $c' \leftarrow_r \mathsf{Enc}^\Pi(\mathsf{pk}_\Pi, \mathsf{RED}^{\mathsf{RSS}\prime}, \vartheta)$ and $(m'_0, \sigma^{\mathsf{SSS}\prime}_0) \leftarrow_r \mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, (m, \sigma^{\mathsf{RSS}}, c, (\tau_i \sigma_i^{\mathsf{SSS}})_{1 \le i \le \ell_m}, \tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}), \sigma_0^{\mathsf{SSS}}, \{(1, m'), (2, \sigma^{\mathsf{RSS}\prime}), (3, c'), (4, (\tau_i, \sigma^{\mathsf{SSS}\prime}i)_{i \in \{1,2,\ldots,\ell\} \setminus \mathsf{MOD}^{\mathsf{PS}}_2})\})$. If $\sigma^{\mathsf{SSS}\prime}_0 = \perp$, return $\perp$. Update $\mathsf{ADM}^{\mathsf{PS}}$ to $\mathsf{ADM}^{\mathsf{PS}\prime}$ by removing all indices in $\mathsf{MOD}^{\mathsf{PS}}_2$ and adjusting the remaining indices by reducing each i in $\mathsf{ADM}^{\mathsf{PS}}$ by $\#\{j \in \mathsf{MOD}^{\mathsf{PS}}_2 : j < i\}$. Return $(\mathsf{MOD}^{\mathsf{PS}}(m), ((\sigma_i^{\mathsf{SSS}})_{i \in \{1,2,\ldots,\ell\} \setminus \mathsf{MOD}^{\mathsf{PS}}_2}, \sigma^{\mathsf{RSS}\prime}, c', \tau, (\tau_i)_{i \in \{1,2,\ldots,\ell\} \setminus \mathsf{MOD}^{\mathsf{PS}}_2}), \mathsf{ADM}^{\mathsf{PS}\prime})$.

$\mathsf{Proof}^{\mathsf{PS}}(\mathsf{sk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \sigma^{\mathsf{PS}}, \{(\sigma_i^{\mathsf{PS}}, m^i)\})$. If for any $(\sigma_i^{\mathsf{PS}}, m_i)$, $0 = \mathsf{Verify}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m_i, \sigma_i^{\mathsf{PS}})$, return $\perp$. If $0 = \mathsf{Verify}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \sigma^{\mathsf{PS}})$, return $\perp$. Return $\mathsf{Proof}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m', \sigma^{\mathsf{SSS}}, \{(\sigma_i^{\mathsf{SSS}}, m'_i)\})$, where $m' = (m, \sigma^{\mathsf{RSS}}, c, (\tau_i)_{1 \le i \le \ell_m}, \tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$ and each $m'_i = (m_i, \sigma_i^{\mathsf{RSS}}, c_i, (\tau_{i,j})_{1 \le j \le \ell_{m_i}}, \tau_i, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$.

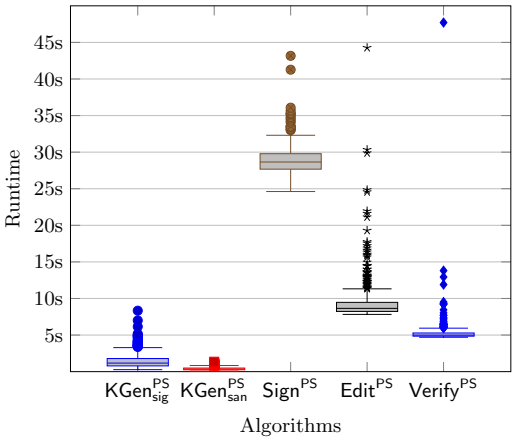
$\mathsf{Judge}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \sigma^{\mathsf{PS}}, \pi^{\mathsf{PS}})$. If $0 = \mathsf{Verify}^{\mathsf{PS}}(\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}, m, \sigma^{\mathsf{PS}})$, return $\perp$. Return $\mathsf{Judge}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m', \sigma^{\mathsf{SSS}}, \pi^{\mathsf{PS}})$, where $m' = (m, \sigma^{\mathsf{RSS}}, c, (\tau_i)_{1 \le i \le \ell_m}, \tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$.

Construction 1: Our PS scheme

Notes on the Construction. We stress that one does not strictly require to use an SSS for a block which is not admissible at signing, but could use a standard signature scheme instead. However, this was not done for two reasons. First, this would further blow up the description. Second, we achieve a stronger form of privacy, i.e., invisibility, if the underlying SSS supports this notion.

Moreover, one may notice that the inner SSSs do not need to be accountable, as the outer SSS is used for this purpose. However, as there are (for obvious reasons) no notions of non-accountable SSSs, there is no way around this, while defining such a notion is not the goal of this paper. Moreover, this can be turned into a useful additional feature, i.e., one can actually generate a proof which blocks have been altered, based on the ideas by Brzuska et al. [12]. How this can be achieved is discussed in Sect. 5.

Implementation. To demonstrate that our scheme is realizable, it was implemented in Java. We chose to implement the version giving the most privacy guarantees, i.e., with invisibility. This was done to show the absolute lower bound of our construction. We discuss this in more depth at the end of the this section. We use the implementation by Beck et al. [3] as the underlying SSS, also using their chameleon-hash with 2,048 Bit RSA moduli and the RSS presented in App. C with 2,048 Bit RSA-FDH signatures, based on the ideas by Brzuska et al. [8]. The security parameter κ is fixed to 512. Furthermore, we stress that the implemented RSS does not have any redaction information ($\mathsf{RED}^{\mathsf{RSS}} = \emptyset$), and thus c is always $\perp$. The measurements were performed on a Lenovo W530 with an Intel i7-3470QM@2.70Ghz, and 16GiB of RAM. No performance optimization such as CRT were implemented and only a single thread does the computations. We evaluated our implementation with 32 blocks, wheres 25% were marked as admissible and an additional 25% as redactable. For editing, 50% of the admissible blocks were sanitized and redacted. We omit proof generation and the judge, as they are simple database look-ups, and parameter generation as it is a one-time setup. The overall results are depicted in Fig. 15a, and Tab. 15b and are based on 500 runs, while verification was measured after sanitization.



(a) Box-plots of the run-times in s

	KGen$^{PS}_{sig}$	KGen$^{PS}_{san}$	Sign^{PS}	Edit^{PS}	Verify^{PS}
Min.:	275	76	24'619	7'825	4'701
25th PCTL:	784	260	27'672	8'230	4'847
Median:	1'172	353	28'655	8'650	4'993
75th PCTL:	1'795	494	29'788	9'475	5'287
90th PCTL:	2'698	650	31'009	11'361	5'918
95th PCTL:	3'464	787	31'842	13'653	6'453
Max.:	8'332	1'356	43'164	44'273	47'713
Average:	1'450	394	28'655	10'937	5'333
SD:	1'004	188	1'938	2'974	2'095

(b) Percentiles for our implementation in ms

Fig. 15: Performance Evaluation Results

As demonstrated, signing is the most expensive operation. Moreover, as already explained, we chose to implement the most expensive version, i.e., the one with invisibility. As this involves rather expensive primitives, i.e., the SSS presented by Beck et al. [3], it is "obvious" that the runtime is not very satisfactory. However, we chose to do this to present a lower bound. In other words, if one is willing to drop invisibility and deploys a weaker SSS (e.g., Brzuska et al [9]; Runtimes are given by de Meer et al. [18]), our resulting construction becomes much more efficient.

5 Extensions and Future Directions

We now present some additional alterations to our construction, further increasing the applicability of the new primitive.

Editing to Signer-Chosen Values. In our basic construction, the editor can either redact a block or edit the block's content to an arbitrary bit-string. However, as already clarified in existing work on SSSs, a signer may want to limit the sanitizer to a set of signer-chosen values [15, 22, 33].

In our construction this extension can easily be integrated, as we bridge the primitives using unique tags and do not further process any output of the inner SSSs. Thus, SSSs supporting this additional feature can directly be used instead of the "standard" SSSs. However, we leave it as future work to derive a formal security model supporting this additional feature.

Block-Level Definitions. Our accountability definition only allows to decide whether a given signature/message pair (σ, m) was altered as a whole. However, as already discussed in prior work, one may also want to decide for each block whether it has been sanitized or not [12,18]. As our construction already uses accountable SSSs as the inner signatures, one can use the inner SSSs' accountability algorithms to create proofs for each block.

Dependencies. In the used definition of the RSS used, the signer can only decide whether a block can be redacted (and/or edited) or not. However, there are use-cases where one wants to force the redactor two follow restrictions, e.g., "if block 1 is redacted, also redact block 2, but block 2 can be redacted on its own" [41]. Due to the black-box usage of the RSS, any RSS which supports this feature can be integrated into our construction. However, we leave a formal definition of this possibility as future work.

Unlinkability. As an additional privacy guarantee, unlinkability prohibits that an adversary can decide which signature was used to create a derived version [11,13,24,36]. However, it seems to be hard to achieve with our underlying construction paradigm.

6 Conclusion

We have introduced the notion of Protean Signature schemes (PS). This new primitive allows to modify and redact signer-chosen parts of a signed message without additional involvement of the signer. We proposed a formal security model and a provably secure black-box construction from sanitizable (SSS) and redactable signature schemes (RSS). Our new primitive generalizes both SSSs and RSSs, while our performance estimates show that our corresponding construction can be considered efficient. However, it remains an open question whether we can construct "designated redactor" RSSs which hide which parts of a message are redactable to achieve a stronger invisibility notion, hiding which blocks are redactable. Likewise, it remains an open question if we can add unlinkability.

References

1. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, a. shelat, and B. Waters. Computing on authenticated data. *J. Cryptology*, 28(2):351–395, 2015.
2. G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS*, pages 159–177, 2005.
3. M. T. Beck, J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Practical strongly invisible and strongly accountable sanitizable signatures. In *ACISP, Part I*, pages 437–452, 2017.
4. A. Bilzhause, M. Huber, H. C. Pöhls, and K. Samelin. Cryptographically enforced four-eyes principle. In *Ares*, pages 760–767, 2016.
5. A. Bilzhause, H. C. Pöhls, and K. Samelin. Position paper: The past, present, and future of sanitizable and redactable signatures. In *Ares*, pages 87:1–87:9, 2017.
6. D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *EuroCrypt*, pages 149–168, 2011.
7. E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. In *PKC*, pages 501–519, 2014.
8. C. Brzuska, H. Busch, Ö. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. Redactable signatures for tree-structured data: Definitions and constructions. In *ACNS*, pages 87–104, 2010.
9. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of sanitizable signatures revisited. In *PKC*, pages 317–336, 2009.

10. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Santizable signatures: How to partially delegate control for authenticated data. In *BIOSIG*, 2009.
11. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of sanitizable signatures. In *PKC*, pages 444–461, 2010.
12. C. Brzuska, H. C. Pöhls, and K. Samelin. Non-interactive public accountability for sanitizable signatures. In *EuroPKI*, pages 178–193, 2012.
13. C. Brzuska, H. C. Pöhls, and K. Samelin. Efficient and perfectly unlinkable sanitizable signatures without group signatures. In *EuroPKI*, pages 12–30, 2013.
14. J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures. In *PKC, Part II*, pages 152–182, 2017.
15. S. Canard and A. Jambert. On extended sanitizable signature schemes. In *CT-RSA*, pages 179–194, 2010.
16. S. Canard, A. Jambert, and R. Lescuyer. Sanitizable signatures with several signers and sanitizers. In *AfricaCrypt*, pages 35–52, 2012.
17. S. Canard, F. Laguillaumie, and M. Milhau. Trapdoorsanitizable signatures and their application to content protection. In *ACNS*, pages 258–276, 2008.
18. 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.
19. H. de Meer, H. C. Pöhls, J. Posegga, and K. Samelin. On the relation between redactable and sanitizable signature schemes. In *ESSOS*, pages 113–130, 2014.
20. D. Demirel, D. Derler, C. Hanser, H. C. Pöhls, D. Slamanig, and G. Traverso. PRISMACLOUD D4.4: Overview of Functional and Malleable Signature Schemes. Technical report, H2020 Prismacloud, www.prismacloud.eu, 2015.
21. D. Derler, H. C. Pöhls, K. Samelin, and D. Slamanig. A general framework for redactable signatures and new constructions. In *ICISC*, pages 3–19, 2015.
22. D. Derler and D. Slamanig. Rethinking privacy for extended sanitizable signatures and a black-box construction of strongly private schemes. In *ProvSec*, 2015.
23. M. Fischlin and P. Harasser. Invisible sanitizable signatures and public-key encryption are equivalent. In *ACNS*, pages 202–220, 2018.
24. N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schröder, and M. Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In *PKC Part-I*, pages 301–330, 2016.
25. E. Ghosh, M. T. Goodrich, O. Ohrimenko, and R. Tamassia. Verifiable zero-knowledge order queries and updates for fully dynamic lists and trees. In *SCN*, pages 216–236, 2016.
26. E. Ghosh, O. Ohrimenko, and R. Tamassia. Zero-knowledge authenticated order queries and order statistics on a list. In *ACNS*, pages 149–171, 2015.
27. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
28. J. Gong, H. Qian, and Y. Zhou. Fully-secure and practical sanitizable signatures. In *Inscrypt*, pages 300–317, 2010.
29. S. Haber, Y. Hatano, Y. Honda, W. G. Horne, K. Miyazaki, T. Sander, S. Tezoku, and D. Yao. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In *AsiaCCS*, pages 353–362, 2008.
30. T. Izu, N. Kunihiro, K. Ohta, M. Sano, and M. Takenaka. Sanitizable and deletable signature. In *WISA*, pages 130–144, 2008.
31. T. Izu, N. Kunihiro, K. Ohta, M. Sano, and M. Takenaka. Yet another sanitizable signature from bilinear maps. In *Ares*, pages 941–946, 2009.
32. R. Johnson, D. Molnar, D. Xiaodong Song, and D. A. Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262, 2002.
33. M. Klonowski and A. Lauks. Extended sanitizable signatures. In *ICISC*, pages 343–355, 2006.
34. S. Krenn, K. Samelin, and D. Sommer. Stronger security for sanitizable signatures. In *DPM/QASA*, pages 100–117, 2015.
35. A. Kundu and E. Bertino. Privacy-preserving authentication of trees and graphs. *Int. J. Inf. Sec.*, 12(6):467–494, 2013.
36. R. W. F. Lai, T. Zhang, S. S. M. Chow, and D. Schröder. Efficient sanitizable signatures without random oracles. In *ESORICS Part-I*, pages 363–380, 2016.
37. K. Miyazaki, M. Iwamura, T. Matsumoto, R. Sasaki, H. Yoshiura, S. Tezuka, and H. Imai. Digitally signed document sanitizing scheme with disclosure condition control. *IEICE Transactions*, 88-A(1):239–246, 2005.
38. H. C. Pöhls and K. Samelin. Accountable redactable signatures. In *Ares*, pages 60–69, 2015.
39. H. C. Pöhls, K. Samelin, and J. Posegga. Sanitizable signatures in XML signature - performance, mixing properties, and revisiting the property of transparency. In *ACNS*, pages 166–182, 2011.
40. K. Samelin, H. C. Pöhls, A. Bilzhause, J. Posegga, and H. de Meer. Redactable signatures for independent removal of structure and content. In *ISPEC*, pages 17–33, 2012.
41. D. Slamanig and S. Rass. Generalizations and extensions of redactable signatures with applications to electronic healthcare. In *CMS*, pages 201–213, 2010.
42. R. Steinfeld, L. Bull, and Y. Zheng. Content extraction signatures. In *ICISC*, pages 285–304, 2001.
43. L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

44. G. Traverso, D. Demirel, and J. A. Buchmann. *Homomorphic Signature Schemes - A Survey*. Springer Briefs in Computer Science. Springer, 2016.
45. R. Tsabary. An equivalence between attribute-based signatures and homomorphic signatures, and new constructions for both. In *TCC*, pages 489–518, 2017.
46. Z. Y. Wu, C.-W. Hsueh, C.-Y. Tsai, F. Lai, H.-C. Lee, and Y.-F. Chung. Redactable signatures for signed CDA documents. *J. Medical Systems*, 36(3):1795–1808, 2012.
47. D. H. Yum, J. W. Seo, and P. Joong Lee. Trapdoor sanitizable signatures made easy. In *ACNS*, pages 53–68, 2010.

A Labeled Public-Key Encryption Schemes

Labeled public-key encryption allows to encrypt a message m using a given public key pk_Π and label $\vartheta \in \{0,1\}^*$. In a nutshell, the given ciphertext leaks no information about the contained message, except its length, if the corresponding secret key sk_Π is not known.

Definition 12 (Labeled Public-Key Encryption Schemes). *A labeled public-key encryption scheme Π consists of four algorithms $\{\mathsf{PPGen}^\Pi, \mathsf{KGen}^\Pi, \mathsf{Enc}^\Pi, \mathsf{Dec}^\Pi\}$, such that:*

PPGen^Π**.** *The algorithm PPGen^Π outputs the public parameters of the scheme:*

$$\mathsf{pp}_\Pi \leftarrow_r \mathsf{PPGen}^\Pi(1^\kappa)$$

It is assumed that pp_Π is implicit input to all other algorithms.

KGen^Π**.** *The algorithm KGen^Π outputs the key pair, on input pp_Π:*

$$(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) \leftarrow_r \mathsf{KGen}^\Pi(\mathsf{pp}_\Pi)$$

Enc^Π**.** *The algorithm Enc^Π gets as input the public key pk_Π, the message $m \in \mathcal{MS}$ and a label $\vartheta \in \{0,1\}^*$ to encrypt. It outputs a ciphertext c:*

$$c \leftarrow_r \mathsf{Enc}^\Pi(\mathsf{pk}_\Pi, m, \vartheta)$$

Dec^Π**.** *The deterministic algorithm Dec^Π outputs a message m (or $\perp$, if the ciphertext is invalid) on input sk_Π, ϑ and a ciphertext c:*

$$m \leftarrow \mathsf{Dec}^\Pi(\mathsf{sk}_\Pi, c, \vartheta)$$

Π IND-CCA2-Security. We need IND-CCA2-security for our construction to work.

Definition 13 (IND-CCA2-Security). *A labeled encryption scheme Π is IND-CCA2-secure, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\left|\Pr[\mathsf{IND\text{-}CCA2}^\Pi_\mathcal{A}(\kappa) = 1] - \tfrac{1}{2}\right| \leq \nu(\kappa)$$

The corresponding experiment is depicted in Figure 16.

B Security Games for SSSs and RSSs

SSSs Security Definitions. We now introduce the security properties required. These are the ones given by Beck et al. [3], but altered for the used notation, already incorporating the strong definitions by Krenn et al. [34]. Moreover, we do neither consider unlinkability, invisibility nor non-interactive public-accountability, as it depends on the context whether these properties are required [3, 14]. However, non-interactive public-accountability is easy to achieve, e.g., by signing the resulting signature again [12], while we discuss unlinkability and invisibility in Sect. 5.

Experiment $\mathsf{IND\text{-}CCA2}^{\Pi}_{\mathcal{A}}(\kappa)$:
$\mathsf{pp}_{\Pi} \leftarrow_r \mathsf{PPGen}^{\Pi}(1^{\kappa})$
$(\mathsf{sk}_{\Pi}, \mathsf{pk}_{\Pi}) \leftarrow_r \mathsf{KGen}^{\Pi}(\mathsf{pp}_{\Pi})$
$b \leftarrow_r \{0,1\}$
$(m_0^*, m_1^*, \vartheta^*, state_{\mathcal{A}}) \leftarrow_r \mathcal{A}^{\mathsf{Dec}^{\Pi}(\mathsf{sk}_{\Pi},\cdot,\cdot)}(\mathsf{pk}_{\Pi})$
If $|m_0^*| \neq |m_1^*| \vee m_0^* \notin \mathcal{MS} \vee m_1^* \notin \mathcal{MS}$:
$c^* \leftarrow \perp$
Else:
$c^* \leftarrow_r \mathsf{Enc}^{\Pi}(\mathsf{pk}_{\Pi}, m_b^*, \vartheta^*)$
$a \leftarrow_r \mathcal{A}^{\mathsf{Dec}^{\Pi\prime}(\mathsf{sk}_{\Pi},\cdot,\cdot)}(state_{\mathcal{A}}, c^*)$
where $\mathsf{Dec}^{\Pi\prime}(\mathsf{sk}_{\Pi},\cdot)$ behaves as $\mathsf{Dec}^{\Pi}(\mathsf{sk}_{\Pi},\cdot)$, but returns $\perp$, if (c^*, ϑ^*) is queried.
return 1, if $a = b$
return 0

Fig. 16: Π IND-CCA2-Security

Experiment $\mathsf{Unforgeability}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{SSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{SSS}}(1^{\kappa})$
$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{SSS}})$
$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{SSS}})$
$(m^*, \sigma^{\mathsf{SSS}*}) \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}},\cdot,\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}})$
for $i = 1, 2, \ldots, q$ let $(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san},i}, m_i, \mathsf{ADM}^{\mathsf{SSS}}_i)$ and σ^{SSS}_i
index the queries/answers to/from $\mathsf{Sign}^{\mathsf{SSS}}$
for $j = 1, 2, \ldots, q'$ let $(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig},j}, m_j, \sigma^{\mathsf{SSS}}_j, \mathsf{MOD}_j)$ and $(m'_j, \sigma^{\mathsf{SSS}\prime}_j)$
index the queries/answers to/from $\mathsf{Sanitize}^{\mathsf{SSS}}$
return 1, if $\mathsf{Verify}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m^*, \sigma^{\mathsf{SSS}*}) = 1 \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m^*, \sigma^{\mathsf{SSS}*}) \neq (\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san},i}, m_i, \sigma^{\mathsf{SSS}}_i) \wedge$
$\forall j \in \{1, 2, \ldots, q'\} : (\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, m^*, \sigma^{\mathsf{SSS}*}) \neq (\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig},j}, m'_j, \sigma^{\mathsf{SSS}\prime}_j)$
return 0

Fig. 17: SSS Unforgeability

Unforgeability. This definition requires that an adversary $\mathcal{A}$ not having any secret keys is not able to produce any new valid signature σ^* on a message m^* which it has never seen, even if $\mathcal{A}$ has full oracle access.

Definition 14 (Unforgeability). *An* SSS *is unforgeable, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*

$$\Pr[\mathsf{Unforgeability}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa) = 1] \leq \nu(\kappa)\,,$$

where the corresponding experiment is defined in Fig. 17.

Immutability. This definition prohibits that an adversary $\mathcal{A}$ can generate a verifying signature $\sigma^{\mathsf{SSS}*}$ for a message m^* not derivable from the signatures given by an honest signer, even if it can generate the sanitizer's key pair.

Definition 15 (Immutability). *An* SSS *is immutable, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*

$$\Pr[\mathsf{Immutability}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa) = 1] \leq \nu(\kappa)\,,$$

where the corresponding experiment is defined in Fig. 18.

Experiment $\mathsf{Immutability}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{SSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{SSS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{SSS}})$
$(m^*, \sigma^{\mathsf{SSS}*}, \mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{san}}) \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}})$
for $i = 1, 2, \ldots, q$ let $(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san},i}, m_i, \mathsf{ADM}^{\mathsf{SSS}}_i)$
index the queries to $\mathsf{Sign}^{\mathsf{SSS}}$
return 1, if $\mathsf{Verify}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{san}}, m^*, \sigma^{\mathsf{SSS}*}) = 1 \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{san}} \neq \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san},i} \vee$
$m^* \notin \{\mathsf{MOD}(m_i) \mid \mathsf{MOD} \text{ with } \mathsf{MOD}^{\mathsf{SSS}}(\mathsf{ADM}^{\mathsf{SSS}}_i) = 1\})$
return 0

Fig. 18: SSS Immutability

Experiment $\mathsf{Privacy}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{SSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{SSS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{SSS}})$
$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{SSS}})$
$b \leftarrow_r \{0, 1\}$
$a \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}},\cdot,\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{LoRSan}(\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}},b)}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}})$
where oracle LoRSan on input of
$m_0, m_1, \mathsf{MOD}^{\mathsf{SSS}}_0, \mathsf{MOD}^{\mathsf{SSS}}_1, \mathsf{ADM}^{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, b$
return $\perp$, if $\mathsf{MOD}^{\mathsf{SSS}}_0 \not\prec (m_0, \mathsf{ADM}^{\mathsf{SSS}}) \vee \mathsf{MOD}^{\mathsf{SSS}}_1 \not\prec (m_1, \mathsf{ADM}^{\mathsf{SSS}}) \vee$
$\mathsf{MOD}^{\mathsf{SSS}}_0(m_0) \neq \mathsf{MOD}^{\mathsf{SSS}}_1(m_1) \vee \mathsf{ADM}^{\mathsf{SSS}} \not\prec m_0 \vee \mathsf{ADM}^{\mathsf{SSS}} \not\prec m_1$
let $\sigma^{\mathsf{SSS}} \leftarrow_r \mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m_b, \mathsf{ADM}^{\mathsf{SSS}})$
return $(m', \sigma^{\mathsf{SSS}\prime}) \leftarrow_r \mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, m_b, \sigma^{\mathsf{SSS}}, \mathsf{MOD}^{\mathsf{SSS}}_b)$
return 1, if $a = b$
return 0

Fig. 19: SSS Privacy

Privacy. This definition prohibits that an adversary $\mathcal{A}$ can learn anything about sanitized parts. This is similar to the definition of standard encryption schemes.

Definition 16 (Privacy). *An* SSS *is private, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*

$$\left|\Pr[\mathsf{Privacy}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)] - 1/2\right| \leq \nu(\kappa)\,,$$

where the corresponding experiment is defined in Fig. 19.

Transparency. This definition prohibits that an adversary $\mathcal{A}$ does not learn whether a signature σ^{SSS} was generated through $\mathsf{Sign}^{\mathsf{SSS}}$ or $\mathsf{Sanitize}^{\mathsf{SSS}}$.

Definition 17 (Transparency). *An* SSS *is transparent, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*

$$\left|\Pr[\mathsf{Transparency}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)] - 1/2\right| \leq \nu(\kappa)\,,$$

where the corresponding experiment is defined in Fig. 20.

Experiment $\mathsf{Transparency}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)$

$\mathsf{pp}_{\mathsf{SSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{SSS}}(1^\kappa)$

$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{SSS}})$

$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{SSS}})$

$b \leftarrow_r \{0,1\}$

$\mathcal{Q} \leftarrow \emptyset$

$a \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}},\cdot,\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{SSS}\prime}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{Sign/Sanit}(\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}},b)}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}})$

where oracle $\mathsf{Proof}^{\mathsf{SSS}\prime}$ on input of

$\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, m, \sigma^{\mathsf{SSS}}, \{(m_i, \sigma^{\mathsf{SSS}}_i) \mid i \in \mathbb{N}\}$:

return $\perp$, if $\mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{san}} = \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}} \wedge$

$((m, \sigma^{\mathsf{SSS}}) \in \mathcal{Q} \ \vee \ \mathcal{Q} \cap \{(m_i, \sigma^{\mathsf{SSS}}_i)\} \neq \emptyset)$

return $\mathsf{Proof}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{san}}, m, \sigma^{\mathsf{SSS}}, \{(m_i, \sigma^{\mathsf{SSS}}_i)\})$

where oracle $\mathsf{Sign/Sanit}$ on input of

$m, \mathsf{MOD}^{\mathsf{SSS}}, \mathsf{ADM}^{\mathsf{SSS}}, \mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}$:

$\sigma^{\mathsf{SSS}} \leftarrow_r \mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m, \mathsf{ADM}^{\mathsf{SSS}})$

$(m', \sigma^{\mathsf{SSS}\prime}) \leftarrow_r \mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, m, \sigma^{\mathsf{SSS}}, \mathsf{MOD}^{\mathsf{SSS}})$

if $b = 1$:

$\sigma^{\mathsf{SSS}\prime} \leftarrow_r \mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m', \mathsf{ADM}^{\mathsf{SSS}})$

If $\sigma^{\mathsf{SSS}\prime} \neq \perp$, set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m', \sigma^{\mathsf{SSS}\prime})\}$

return $(m', \sigma^{\mathsf{SSS}\prime})$

return 1, if $a = b$

return 0

Fig. 20: SSS Transparency

Experiment $\mathsf{SigAccountability}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)$

$\mathsf{pp}_{\mathsf{SSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{SSS}}(1^\kappa)$

$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{SSS}})$

$(\mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{sig}}, \pi^{\mathsf{SSS}*}, m^*, \sigma^{\mathsf{SSS}*}) \leftarrow_r \mathcal{A}^{\mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}})$

for $i = 1, 2, \ldots, q$ let $(m'_i, \sigma^{\mathsf{SSS}\prime}_i)$ and $(m_i, \mathsf{MOD}^{\mathsf{SSS}}_i, \sigma^{\mathsf{SSS}}_i, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig},i})$

index the answers/queries from/to $\mathsf{Sanitize}^{\mathsf{SSS}}$

return 1, if $\mathsf{Verify}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m^*, \sigma^{\mathsf{SSS}*}) = 1 \wedge$

$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{sig}}, m^*, \sigma^{\mathsf{SSS}*}) \neq (\mathsf{pk}_{\mathsf{sig},i}, m'_i, \sigma^{\mathsf{SSS}\prime}_i) \wedge$

$\mathsf{Judge}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m^*, \sigma^{\mathsf{SSS}*}, \pi^{\mathsf{SSS}*}) = \mathsf{San}^{\mathsf{SSS}}$

return 0

Fig. 21: SSS Signer-Accountability

Signer-Accountability. Signer-accountability prohibits that an adversary can generate a bogus proof that makes $\mathsf{Judge}^{\mathsf{SSS}}$ decide that the sanitizer is responsible for a given signature/message pair $(m^*, \sigma^{\mathsf{SSS}*})$, but the sanitizer has never generated this pair. This is even true, if the adversary can generate the signer's key pair.

Definition 18 (Signer-Accountability). *An* SSS *is signer-accountable, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*

$$\Pr[\mathsf{SigAccountability}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa) = 1] \leq \nu(\kappa)\,,$$

where the corresponding experiment is defined in Fig. 21.

Experiment $\mathsf{SanAccountability}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{SSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{SSS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{SSS}})$
$(m^*, \sigma^{\mathsf{SSS}*}, \mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{san}}) \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}})$
for $i = 1, 2, \ldots, q$ let $(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san},i}, m_i, \mathsf{ADM}^{\mathsf{SSS}}_i)$ and σ^{SSS}_i
index the queries/answers to/from $\mathsf{Sign}^{\mathsf{SSS}}$
$\pi^{\mathsf{SSS}} \leftarrow_r \mathsf{Proof}^{\mathsf{SSS}}(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{san}}, m^*, \sigma^{\mathsf{SSS}*}, \{(m_i, \sigma^{\mathsf{SSS}}_i) \mid 0 < i \leq q\})$
return 1, if $\mathsf{Verify}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{san}}, m^*, \sigma^{\mathsf{SSS}*}) = 1 \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{san}}, m^*, \sigma^{\mathsf{SSS}*}) \neq (\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san},i}, m_i, \sigma^{\mathsf{SSS}}_i) \wedge$
$\mathsf{Judge}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}*}_{\mathsf{san}}, m^*, \sigma^{\mathsf{SSS}*}, \pi^{\mathsf{SSS}}) = \mathsf{Sig}^{\mathsf{SSS}}$
return 0

Fig. 22: SSS Sanitizer-Accountability

Sanitizer-Accountability. Sanitizer-accountability prohibits that an adversary can generate a bogus signature/message pair $(m^*, \sigma^{\mathsf{SSS}*})$ that makes $\mathsf{Proof}^{\mathsf{SSS}}$ outputs a (honestly generated) generated proof π^{SSS} which points to the signer, but $(m^*, \sigma^{\mathsf{SSS}*})$ has never been generated by the signer. This is even true, if the adversary can generate the sanitizer's key pair.

Definition 19 (Sanitizer-Accountability). *An* SSS *is sanitizer-accountable, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that*

$$\Pr[\mathsf{SanAccountability}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa) = 1] \leq \nu(\kappa)\,,$$

where the corresponding experiment is defined in Fig. 22.

Invisibility. Depending on the context, an additional privacy guarantee may be required. In particular, invisibility prohibits that an outsider holding no secret keys can decide which parts of a message m are sanitizable. Note, the signing oracle can be simulated using the LoRADM oracle and setting $\mathsf{ADM}^{\mathsf{SSS}}_0 = \mathsf{ADM}^{\mathsf{SSS}}_1$. The notation $\mathsf{ADM}^{\mathsf{SSS}}_0 \cap \mathsf{ADM}^{\mathsf{SSS}}_1$ means that only those indices are admissible which are admissible in $\mathsf{ADM}^{\mathsf{SSS}}_0$ and $\mathsf{ADM}^{\mathsf{SSS}}_1$.

Definition 20 (Invisibility). *An* SSS *is invisible, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that*

$$\left|\Pr[\mathsf{Invisibility}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)] - 1/2\right| \leq \nu(\kappa)\,,$$

where the corresponding experiment is defined in Fig. 23.

RSSs Security Definitions. We now introduce the security model for RSSs. This is a simplified version derived from Derler et al. [21], which is, in turn, derived from Brzuska et al. [8]. Note, moreover, that we do not need accountability, as in our construction accountability is given by the SSS, much like Pöhls and Samelin [38] and Bilzhause et al. [4].

Unforgeability. This definition requires that an adversary $\mathcal{A}$ cannot derive a message which is not derivable from any signed messages. We stress that, even though the set $\bigcup_{i=1}^{q}\{\mathsf{MOD}^{\mathsf{RSS}}(m_i) \mid \mathsf{MOD}^{\mathsf{RSS}} \prec (m_i, \mathsf{ADM}^{\mathsf{RSS}}_i)\}$ may grow exponentially, membership is trivially to decide, i.e., in polynomial time.

Definition 21 (Unforgeability). *An* RSS *is unforgeable, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that*

$$\Pr[\mathsf{Unforgeability}^{\mathsf{RSS}}_{\mathcal{A}}(\kappa) = 1] \leq \nu(\kappa)\,,$$

where the corresponding experiment is defined in Fig. 24.

Experiment $\mathsf{Invisibility}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)$

$\mathsf{pp}_{\mathsf{SSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{SSS}}(1^\kappa)$

$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{SSS}})$

$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{SSS}})$

$b \leftarrow_r \{0,1\}$

$\mathcal{Q} \leftarrow \emptyset$

$a \leftarrow_r \mathcal{A}^{\mathsf{Sanitize}^{\mathsf{SSS}\prime}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}},\cdot,\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{SSS}\prime}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{LoRADM}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot,b)}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}})$

where oracle LoRADM on input of $\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{san}}, m, \mathsf{ADM}^{\mathsf{SSS}}_0, \mathsf{ADM}^{\mathsf{SSS}}_1, b$:

return $\bot$, if $\mathsf{ADM}^{\mathsf{SSS}}_0 \prec m \wedge \mathsf{ADM}^{\mathsf{SSS}}_1 \prec m$

return $\bot$, if $\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}} \neq \mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{san}} \ \wedge \ \mathsf{ADM}^{\mathsf{SSS}}_0 \neq \mathsf{ADM}^{\mathsf{SSS}}_1$

let $\sigma^{\mathsf{SSS}} \leftarrow_r \mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{san}}, m, \mathsf{ADM}^{\mathsf{SSS}}_b)$

if $\mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{san}} = \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}$, let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m, \sigma^{\mathsf{SSS}}, \mathsf{ADM}^{\mathsf{SSS}}_0 \cap \mathsf{ADM}^{\mathsf{SSS}}_1)\}$

return σ^{SSS}

where oracle $\mathsf{Sanitize}^{\mathsf{SSS}\prime}$ on input of $\mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, m, \mathsf{MOD}^{\mathsf{SSS}}, \sigma^{\mathsf{SSS}}$:

return $\bot$, if $\mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{sig}} = \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}} \wedge \nexists (m, \sigma^{\mathsf{SSS}}, \mathsf{ADM}^{\mathsf{SSS}}) \in \mathcal{Q} : \mathsf{MOD}^{\mathsf{SSS}} \prec (m, \mathsf{ADM}^{\mathsf{SSS}})$

let $(m', \sigma^{\mathsf{SSS}\prime}) \leftarrow_r \mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, m, \mathsf{MOD}^{\mathsf{SSS}}, \sigma^{\mathsf{SSS}})$

if $\mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{sig}} = \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}} \wedge \exists (m, \sigma^{\mathsf{SSS}}, \mathsf{ADM}^{\mathsf{SSS}\prime}) \in \mathcal{Q} : \mathsf{MOD}^{\mathsf{SSS}} \prec (m, \mathsf{ADM}^{\mathsf{SSS}\prime})$,

let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m', \sigma^{\mathsf{SSS}\prime}, \mathsf{ADM}^{\mathsf{SSS}\prime})\}$

return $(m', \sigma^{\mathsf{SSS}\prime})$

return 1, if $a = b$

return 0

Fig. 23: SSS Invisibility

Experiment $\mathsf{Unforgeability}^{\mathsf{RSS}}_{\mathcal{A}}(\kappa)$

$\mathsf{pp}_{\mathsf{RSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{RSS}}(1^\kappa)$

$(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{RSS}}(\mathsf{pp}_{\mathsf{RSS}})$

$(m^*, \sigma^{\mathsf{RSS}*}) \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{RSS}}(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}},\cdot,\cdot)}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}})$

for $i = 1, 2, \ldots, q$ let m_i, $\mathsf{ADM}^{\mathsf{RSS}}_i$, and σ^{RSS}_i index the queries/answers to/from Sign

return 1, if $\mathsf{Verify}^{\mathsf{RSS}}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}, m^*, \sigma^{\mathsf{RSS}*}) = 1 \wedge$

$m^* \notin \bigcup_{i=1}^{q} \{\mathsf{MOD}^{\mathsf{RSS}}(m_i) \mid \mathsf{MOD}^{\mathsf{RSS}} \prec (m_i, \mathsf{ADM}^{\mathsf{RSS}}_i)\}$

return 0

Fig. 24: RSS Unforgeability

Privacy. This definition prohibits that an adversary $\mathcal{A}$ can learn anything about redacted parts. This is similar to the definition for SSSs.

Definition 22 (Privacy). *An* RSS *is private, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*

$$\left|\Pr[\mathsf{Privacy}^{\mathsf{RSS}}_{\mathcal{A}}(\kappa)] - 1/2\right| \leq \nu(\kappa)\,,$$

where the corresponding experiment is defined in Fig. 25.

Transparency. This definition prohibits that an adversary $\mathcal{A}$ can decide whether a signature was generated through $\mathsf{Sign}^{\mathsf{RSS}}$

Definition 23 (Transparency). *An* RSS *is transparent, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*

$$\left|\Pr[\mathsf{Transparency}^{\mathsf{RSS}}_{\mathcal{A}}(\kappa)] - 1/2\right| \leq \nu(\kappa)\,,$$

Experiment $\mathsf{Privacy}^{\mathsf{RSS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{RSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{RSS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{RSS}}(\mathsf{pp}_{\mathsf{RSS}})$
$b \leftarrow_r \{0,1\}$
$a \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{RSS}}(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}},\cdot,\cdot),\mathsf{LoRRedact}(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot,b)}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}})$
where oracle $\mathsf{LoRRedact}$ on input of
$\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, m_0, m_1, \mathsf{MOD}^{\mathsf{RSS}}_0, \mathsf{MOD}^{\mathsf{RSS}}_1, b$
let $(\sigma^{\mathsf{RSS}}_0, \mathsf{RED}^{\mathsf{RSS}}_0) \leftarrow_r \mathsf{Sign}^{\mathsf{RSS}}(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, m_0)$
let $(\sigma^{\mathsf{RSS}}_1, \mathsf{RED}^{\mathsf{RSS}}_1) \leftarrow_r \mathsf{Sign}^{\mathsf{RSS}}(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, m_1)$
let $(\sigma^{\mathsf{RSS}\prime}_0, m'_0, \mathsf{RED}^{\mathsf{RSS}\prime}_0) \leftarrow_r \mathsf{Redact}^{\mathsf{RSS}}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}, m_0, \sigma^{\mathsf{RSS}}_0, \mathsf{MOD}^{\mathsf{RSS}}_0, \mathsf{RED}^{\mathsf{RSS}}_0)$
let $(\sigma^{\mathsf{RSS}\prime}_1, m'_1, \mathsf{RED}^{\mathsf{RSS}\prime}_1) \leftarrow_r \mathsf{Redact}^{\mathsf{RSS}}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}, m_1, \sigma^{\mathsf{RSS}}_1, \mathsf{MOD}^{\mathsf{RSS}}_1, \mathsf{RED}^{\mathsf{RSS}}_1)$
return $\perp$, if $m'_0 \neq m'_1 \ \vee \ \mathsf{ADM}^{\mathsf{RSS}}_0 \neq \mathsf{ADM}^{\mathsf{RSS}}_1$
return $(m'_b, \sigma^{\mathsf{RSS}\prime}_b)$
return 1, if $a = b$
return 0

Fig. 25: RSS Privacy

Experiment $\mathsf{Transparency}^{\mathsf{RSS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{RSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{RSS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{RSS}}(\mathsf{pp}_{\mathsf{RSS}})$
$b \leftarrow_r \{0,1\}$
$a \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{RSS}}(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}},\cdot,\cdot),\mathsf{Sign/Redact}(\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}},b)}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}})$
where oracle $\mathsf{Sign/Redact}$ on input of
$m, \mathsf{MOD}^{\mathsf{RSS}}, \mathsf{ADM}^{\mathsf{RSS}}, \mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, b$:
$(\sigma^{\mathsf{RSS}}, \mathsf{RED}^{\mathsf{RSS}}) \leftarrow_r \mathsf{Sign}^{\mathsf{RSS}}(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, m, \mathsf{ADM}^{\mathsf{RSS}})$
$(m', \sigma^{\mathsf{RSS}\prime}, \mathsf{RED}^{\mathsf{RSS}\prime}) \leftarrow_r \mathsf{Redact}^{\mathsf{RSS}}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}, m, \sigma^{\mathsf{RSS}}, \mathsf{MOD}^{\mathsf{RSS}}, \mathsf{RED}^{\mathsf{RSS}})$
if $b = 1$:
$(\sigma^{\mathsf{RSS}\prime}, \mathsf{RED}^{\mathsf{RSS}\prime}) \leftarrow_r \mathsf{Sign}^{\mathsf{RSS}}(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, m', \mathsf{ADM}^{\mathsf{RSS}\prime})$
return $(m', \sigma^{\mathsf{RSS}\prime})$
return 1, if $a = b$
return 0

Fig. 26: RSS Transparency

where the corresponding experiment is defined in Fig. 26.

C RSS With Redaction Control

We now show how to construct an RSS′ which allows prohibiting redactions in a black-box fashion from any existing secure RSS as defined. Note, this construction is not unlinkable, even though the underlying RSS may be.

The basic idea is as follows. The message m is extended to contain an additional block m^0 which contains all blocks which are non-redactable. To mark that block non-redactable, it contains a leading 0, while all other blocks contain a leading 1. To bind everything together, each block $m^i \in m$ (and m^0) is bound to an additional tag τ. Redaction works as before, but verification also checks whether the blocks contained in m^0 are also contained in m in the correct order.

$\mathsf{PPGen}^{\mathsf{RSS}\prime}(1^\kappa)$. Return $\mathsf{pp}_{\mathsf{RSS}}{}' = \mathsf{RSS.PPGen}^{\mathsf{RSS}}(1^\kappa)$.
$\mathsf{KGen}^{\mathsf{RSS}\prime}(\mathsf{pp}_{\mathsf{RSS}})$. Return $(\mathsf{sk}_{\mathsf{sig}}^{\mathsf{RSS}\prime}, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}\prime}) = \mathsf{RSS.KGen}^{\mathsf{RSS}}(\mathsf{pp}_{\mathsf{RSS}}{}')$.

$\mathsf{Sign}^{\mathsf{RSS}\prime}(\mathsf{sk}_{\mathsf{sig}}^{\mathsf{RSS}\prime}, m, \mathsf{ADM}^{\mathsf{RSS}})$. If $\mathsf{ADM}^{\mathsf{RSS}} \prec m$, where $m = (m^1, m^2, \ldots, m^\ell)$, continue. Otherwise, return $\perp$. Draw $\tau_i \leftarrow_r \{0,1\}^\kappa$ for $i \in \{0, 1, \ldots, \ell\}$ Let $m^{0\prime} = (0, \tau, (\tau^i, m^i)_{i \notin \mathsf{ADM}^{\mathsf{RSS}}})$, $m^{i\prime} = (1, \tau, \tau_i, m^i)$ and $m' = (m^{0\prime}, m^{1\prime}, \ldots, m^{\ell\prime})$. Return $(\sigma^{\mathsf{RSS}\prime\prime}, \mathsf{RED}^{\mathsf{RSS}}) = ((\sigma_1^{\mathsf{RSS}}, \sigma_2^{\mathsf{RSS}}, (\tau_i)_{0 \le i \le \ell}), \mathsf{RED}^{\mathsf{RSS}})$, where $(\sigma_1^{\mathsf{RSS}}, \mathsf{RED}^{\mathsf{RSS}}) = \mathsf{RSS.Sign}^{\mathsf{RSS}}(\mathsf{sk}_{\mathsf{sig}}^{\mathsf{RSS}\prime}, m', \emptyset)$ and $\sigma_2^{\mathsf{RSS}} = (\tau_i, m^{i\prime})_{i \notin \mathsf{ADM}^{\mathsf{RSS}}}$.
$\mathsf{Verify}^{\mathsf{RSS}\prime}(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}}, m, \sigma^{\mathsf{RSS}})$. Parse σ^{RSS} as $(\sigma_1^{\mathsf{RSS}}, \sigma_2^{\mathsf{RSS}}, (\tau_i)_{0 \le i \le \ell})$. Let $m^{0\prime} = (0, \tau, (\tau^i, m^i)_{(\tau_i, m^i) \in \sigma_2^{\mathsf{RSS}}})$, $m^{i\prime} = (1, \tau, \tau_i, m^i)$ and $m' = (m^{0\prime}, m^{1\prime}, \ldots, m^{\ell\prime})$. Return $\mathsf{RSS.Verify}^{\mathsf{RSS}}(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}}, m', \sigma_1^{\mathsf{RSS}})$.
$\mathsf{Redact}^{\mathsf{RSS}\prime}(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}}, m, \sigma^{\mathsf{RSS}}, \mathsf{MOD}^{\mathsf{RSS}}, \mathsf{RED}^{\mathsf{RSS}})$. If $\mathsf{Verify}^{\mathsf{RSS}\prime}(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}}, m, \sigma^{\mathsf{RSS}}) = 0$, return $\perp$. Parse σ^{RSS} as $(\sigma_1^{\mathsf{RSS}}, \sigma_2^{\mathsf{RSS}}, (\tau_i)_{0 \le i \le \ell})$. Let $m^{0\prime} = (0, \tau, (\tau^i, m^i)_{(\tau_i, m^i) \in \sigma_2^{\mathsf{RSS}}})$, $m^{i\prime} = (1, \tau, \tau_i, m^i)$ and $m' = (m^{0\prime}, m^{1\prime}, \ldots, m^{\ell\prime})$ and $\mathsf{MOD}^{\mathsf{RSS}\prime}$ is the same as $\mathsf{MOD}^{\mathsf{RSS}}$, but each element is incremented by 1. Let $(\sigma_1^{\mathsf{RSS}\prime}, m'', \mathsf{RED}^{\mathsf{RSS}\prime}) \leftarrow_r \mathsf{Redact}^{\mathsf{RSS}}(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}}, m', \sigma^{\mathsf{RSS}}, \mathsf{MOD}^{\mathsf{RSS}\prime}, \mathsf{RED}^{\mathsf{RSS}})$. Return $((\sigma_1^{\mathsf{RSS}\prime}, \sigma_2^{\mathsf{RSS}}, (\tau_i, m^i)_{0 \le i \le \ell \setminus \mathsf{MOD}^{\mathsf{RSS}\prime}}), \mathsf{MOD}^{\mathsf{RSS}}(m), \mathsf{RED}^{\mathsf{RSS}\prime})$.

Construction 2: Black-Box RSS construction with non-redactable blocks

Let $\mathsf{RSS} = \{\mathsf{PPGen}^{\mathsf{RSS}}, \mathsf{KGen}^{\mathsf{RSS}}, \mathsf{Sign}^{\mathsf{RSS}}, \mathsf{Verify}^{\mathsf{RSS}}, \mathsf{Redact}^{\mathsf{RSS}}\}$ be a secure RSS, which assumes $\mathsf{ADM}^{\mathsf{RSS}} = \emptyset$, i.e., the RSS does not support any redaction control. We now construct $\mathsf{RSS}' = \{\mathsf{PPGen}^{\mathsf{RSS}\prime}, \mathsf{KGen}^{\mathsf{RSS}\prime}, \mathsf{Sign}^{\mathsf{RSS}\prime}, \mathsf{Verify}^{\mathsf{RSS}\prime}, \mathsf{Redact}^{\mathsf{RSS}\prime}\}$ as given in Construction 2, where $\mathsf{ADM}^{\mathsf{RSS}}$ is defined as given in Sect. 2.

Security. We now prove that the above construction is secure.

Theorem 3. *The above construction is secure, if* RSS *is secure and allows to mark blocks as non-redactable.*

We first provide a sketch of the proof, while the full proof is given afterwards.

Proof (Sketch). The proof idea is simple. On the one hand, transparency and privacy directly follow from the underlying RSS and the uniform distribution of the tags. Unforgeability, on the other hand, follows from the fact that the first block in the modified message contains all non-redactable blocks and cannot be removed without detection, while each block is bound to a specific message.

Proof. Correctness follows from inspection. Moreover, as transparency implies privacy (only in the RSS case!), it is sufficient to prove transparency and unforgeability [8, 21]. We prove each property on its own.

Transparency. To prove that our scheme is transparent, we use a sequence of games:

Game 0: The original transparency game, where $b = 0$.
Game 1: We directly sign m' instead of signing and redacting.
Transition - Game 0 → Game 1: If $\mathcal{A}$ can distinguish this hop, it can be turned into an adversary $\mathcal{B}$ against the transparency of the underlying RSS. A reduction is simple. Namely, the reduction $\mathcal{B}$ receives the challenge key $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}\prime}$ from its own challenger, and embeds it as $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}}$. Every underlying signing request for the RSS is performed by the reduction's oracles. So far, the simulation is perfect. Finally, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. Note, τ is distributed identically in all cases.
Thus, $|\Pr[S_0] - \Pr[S_1]| \le \nu_{\mathsf{rss\text{-}trans}}(\kappa)$ follows, while we are now in the case $b = 1$. Moreover, each hop changes the view of the adversary only negligibly, concluding the proof.

Unforgeability. To prove that our scheme is unforgeable, we use a sequence of games:

Game 0: The original unforgeability game.
Game 1: We now abort, if τ was drawn twice.
Transition - Game 0 → Game 1: The probability that this event happens is bounded by the birthday paradox. $|\Pr[S_0] - \Pr[S_1]| \le q_s^2/2^\kappa$ follows, where q_s is the number of drawn tags.
Game 2: We now abort, if the adversary outputs (m^*, σ^*), which was not derivable.

Experiment $\mathsf{Unlinkability}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{SSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{SSS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{SSS}})$
$(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}) \leftarrow_r \mathsf{KGen}^{\mathsf{SSS}}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{SSS}})$
$b \leftarrow_r \{0,1\}$
$a \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}},\cdot,\cdot,\cdot,\cdot),\mathsf{Proof}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{LoRSan}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}},\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}},b)}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}})$
where oracle LoRSan on input of
$m_0, m_1, \mathsf{MOD}^{\mathsf{SSS}}_0, \mathsf{MOD}^{\mathsf{SSS}}_1, \sigma^{\mathsf{SSS}}_0, \sigma^{\mathsf{SSS}}_1, \mathsf{sk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, b$
return $\bot$, if $\mathsf{Verify}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m_0, \sigma^{\mathsf{SSS}}_0) = 0 \ \vee \ \mathsf{Verify}^{\mathsf{SSS}}(\mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}}, m_1, \sigma^{\mathsf{SSS}}_1) = 0 \vee$
$\mathsf{ADM}^{\mathsf{SSS}}_0 \neq \mathsf{ADM}^{\mathsf{SSS}}_1 \ \vee \ \mathsf{ADM}^{\mathsf{SSS}} \not\ni m_0 \ \vee \ \mathsf{ADM}^{\mathsf{SSS}} \not\ni m_1 \ \vee \ \mathsf{MOD}^{\mathsf{SSS}}_0(m_0) \neq \mathsf{MOD}^{\mathsf{SSS}}_1(m_1) \vee$
$\mathsf{MOD}^{\mathsf{SSS}}_0 \not\ni (m_0, \mathsf{ADM}^{\mathsf{SSS}}_0) \ \vee \ \mathsf{MOD}^{\mathsf{SSS}}_1 \not\ni (m_1, \mathsf{ADM}^{\mathsf{SSS}}_1)$
return $(m', \sigma^{\mathsf{SSS}\prime}) \leftarrow_r \mathsf{Sanitize}^{\mathsf{SSS}}(\mathsf{sk}^{\mathsf{SSS}}_{\mathsf{san}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{sig}}, m_b, \mathsf{MOD}^{\mathsf{SSS}}_b)$
return 1, if $a = b$
return 0

Fig. 27: SSS Unlinkability

Transition - Game 0 → Game 1: If $\mathcal{A}$ can generate such a pair, it can be turned into an adversary $\mathcal{B}$ against the unforgeability of the underlying RSS. A reduction is simple. Namely, the reduction $\mathcal{B}$ receives the challenge key $\mathsf{pk}^{\mathsf{RSS}\prime}_{\mathsf{sig}}$ from its own challenger, and embeds it as $\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}$. Every underlying signing request for the RSS is performed by the reduction's oracles. So far, the simulation is perfect. Then, as m^* was not derivable by assumption, either because some block is fresh, the order is not correct or a block was redacted, even though it was marked as non-redactable. In either case, $\mathcal{B}$ can return $(m^{*\prime}, \sigma^{*\prime})$, where $m^{*\prime} = (m^{0\prime}, m^{1\prime}, \ldots, m^{\ell\prime})$, $m^{0\prime} = (0, \tau, (\tau_i, m_i)_{i \notin \mathsf{ADM}^{\mathsf{RSS}}})$, $m^{i\prime} = (1, \tau, m^i)$ and $(\sigma^{*\prime}, \sigma^{\mathsf{RSS}}_2, (\tau_i)_{0 \le i \le \ell \setminus \mathsf{MOD}^{\mathsf{RSS}}})$, as either $m^{0\prime}$ is forged or the other blocks, while the tags τ_i force a block to remain in the signature. Thus, $|\Pr[S_1] - \Pr[S_2]| \le \nu_{\mathsf{rss\text{-}unf}}(\kappa)$ follows.
Now, the adversary has no more possibilities to forge a signature. Moreover, each hop changes the view of the adversary only negligibly, concluding the proof.

D Additional Security Properties

We now give additional security properties, not required to understand the main body of this paper.

Additional Definitions for SSSs.

Unlinkability. This definition prohibits that an adversary $\mathcal{A}$ can learn from which signature a sanitized signature was derived from.

Definition 24 (Unlinkability). *An* SSS *is unlinkable, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that*

$$\left|\Pr[\mathsf{Unlinkability}^{\mathsf{SSS}}_{\mathcal{A}}(\kappa)] - 1/2\right| \le \nu(\kappa)\,,$$

where the corresponding experiment is defined in Fig. 27.

Additional Definitions for RSSs.

Experiment $\mathsf{Unlinkability}^{\mathsf{RSS}}_{\mathcal{A}}(\kappa)$
$\mathsf{pp}_{\mathsf{RSS}} \leftarrow_r \mathsf{PPGen}^{\mathsf{RSS}}(1^\kappa)$
$(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}) \leftarrow_r \mathsf{KGen}^{\mathsf{RSS}}(\mathsf{pp}_{\mathsf{RSS}})$
$b \leftarrow_r \{0,1\}$
$a \leftarrow_r \mathcal{A}^{\mathsf{Sign}^{\mathsf{RSS}}(\mathsf{sk}^{\mathsf{RSS}}_{\mathsf{sig}},\cdot,\cdot),\mathsf{LoRRed}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,b)}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}})$
where oracle LoRRed on input of
$m_0, m_1, \mathsf{MOD}^{\mathsf{RSS}}_0, \mathsf{MOD}^{\mathsf{RSS}}_1, \sigma^{\mathsf{RSS}}_0, \sigma^{\mathsf{RSS}}_1, \mathsf{RED}^{\mathsf{RSS}}_0, \mathsf{RED}^{\mathsf{RSS}}_1, b$
return $\bot$, if $\mathsf{ADM}^{\mathsf{RSS}}_0 \neq \mathsf{ADM}^{\mathsf{RSS}}_1 \ \vee \ \mathsf{ADM}^{\mathsf{RSS}}_0 \not\prec m_0 \vee$
$\mathsf{ADM}^{\mathsf{RSS}}_1 \not\prec m_1 \ \vee \ \mathsf{MOD}^{\mathsf{RSS}}_0(m_0) \neq \mathsf{MOD}^{\mathsf{RSS}}_1(m_1) \vee$
$\mathsf{MOD}^{\mathsf{RSS}}_0 \not\prec (m_0, \mathsf{ADM}^{\mathsf{RSS}}_0) \ \vee \ \mathsf{MOD}^{\mathsf{RSS}}_1 \not\prec (m_1, \mathsf{ADM}^{\mathsf{RSS}}_1)$
let $(m'_0, \sigma^{\mathsf{RSS}\prime}_0, \mathsf{RED}^{\mathsf{RSS}\prime}_0) \leftarrow_r \mathsf{Redact}^{\mathsf{RSS}}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}, m_0, \mathsf{MOD}^{\mathsf{RSS}}_0, \mathsf{RED}^{\mathsf{RSS}}_0)$
let $(m'_1, \sigma^{\mathsf{RSS}\prime}_1, \mathsf{RED}^{\mathsf{RSS}\prime}_1) \leftarrow_r \mathsf{Redact}^{\mathsf{RSS}}(\mathsf{pk}^{\mathsf{RSS}}_{\mathsf{sig}}, m_1, \mathsf{MOD}^{\mathsf{RSS}}_1, \mathsf{RED}^{\mathsf{RSS}}_1)$
return $\bot$, if $\sigma^{\mathsf{RSS}\prime}_0 = \bot \ \vee \ \sigma^{\mathsf{RSS}\prime}_1 = \bot$
return $(m'_b, \sigma^{\mathsf{RSS}\prime}_b, \mathsf{RED}^{\mathsf{RSS}\prime}_b)$
return 1, if $a = b$
return 0

Fig. 28: RSS Unlinkability

Unlinkability. This definition prohibits that an adversary $\mathcal{A}$ learns from which signature a redacted signature was derived from.

Definition 25 (Unlinkability). *An* RSS *is unlinkable, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that*

$$\left|\Pr[\mathsf{Unlinkability}^{\mathsf{RSS}}_{\mathcal{A}}(\kappa)] - 1/2\right| \leq \nu(\kappa)\,,$$

where the corresponding experiment is defined in Fig. 28.

E Proof of Theorem 2

Proof. Correctness follows from inspection. Each security property is proven on it own. However, we already keep all queries and answers to and from the oracle. This does not change the view of the adversary. We also directly generate any keys required, but not received by the reduction's own challenger, honestly, also embedding them, without mentioning it, to shorten the proof.

Unforgeability. To prove that our scheme is unforgeable, we use a sequence of games:

Game 0: The original unforgeability game.

Game 1: We now abort, if the adversary outputs $(m, \sigma^{\mathsf{PS}})$, where $\sigma^{\mathsf{PS}} = ((\sigma^{\mathsf{SSS}}_i)_{0\leq i\leq \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1\leq i\leq \ell_m})$, where any $(m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma^{\mathsf{SSS}}_i)_{1\leq i\leq \ell_m}, \tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$, protected by σ^{SSS}_0, has never been returned by the challenger.

Transition - Game 0 → Game 1: In this case, $((m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma^{\mathsf{SSS}}_i)_{1\leq i\leq \ell_m}, \tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}), \sigma^{\mathsf{SSS}}_0)$ is a valid forgery of the outer SSS. A reduction is simple. Namely, the reduction $\mathcal{B}$ receives the challenge keys $\mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{sig}}$ and $\mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{san}}$ from its own challenger, and embeds them into $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}$ and $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}$. Every underlying signing and sanitization request for the SSSs is performed by the reduction's oracles. As, by assumption, the message protected by σ^{SSS}_0 must be fresh, it thus breaks the unforgeability of the underlying SSS in any case. Thus, $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{sss\text{-}unf}}(\kappa)$ follows.

Game 2: We now abort, if the adversary outputs $(m, \sigma^{\mathsf{PS}})$, where $\sigma^{\mathsf{PS}} = ((\sigma_i^{\mathsf{SSS}})_{0\le i\le \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1\le i\le \ell_m})$, where any $(m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1\le i\le \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$ protected by σ_0^{SSS} was returned by the challenger, but σ_0^{SSS} was never created by the challenger.

Transition - Game 1 → Game 2: In this case, $((m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1\le i\le \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}}), \sigma_0^{\mathsf{SSS}})$ is a valid forgery of the outer SSS. The reduction works as in the prior hop. As, by assumption, the message protected by σ_0^{SSS} must be fresh, it thus breaks the unforgeability of the underlying SSS in any case. Thus, $|\Pr[S_1] - \Pr[S_2]| \le \nu_{\mathsf{sss\text{-}unf}}(\kappa)$ follows.

Now, the adversary can no longer win the unforgeability game, as also each public key is bound to a tag. Moreover, each hop changes the view of the adversary only negligibly, concluding the proof.

Signer-Accountability. To prove that our scheme is signer-accountable, we use a sequence of games:

Game 0: The original signer-accountability game.

Game 1: We now abort, if the adversary outputs $(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \pi^{\mathsf{PS}}, m, \sigma^{\mathsf{PS}})$, where $\sigma^{\mathsf{PS}} = ((\sigma_i^{\mathsf{SSS}})_{0\le i\le \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1\le i\le \ell_m})$, where any $(m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1\le i\le \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$, protected by σ_0^{SSS}, has never been returned by the challenger.

Transition - Game 0 → Game 1: In this case, $(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{SSS}}, \pi^{\mathsf{PS}}, (m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1\le i\le \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}}), \sigma_0^{\mathsf{SSS}})$ is a valid forgery of the outer SSS. For the reduction, $\mathcal{B}$ receives the challenge keys $\mathsf{pk}_{\mathsf{san}}^{\mathsf{SSS}\prime}$ from its own challenger, and embeds them into $\mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}}$. Every underlying sanitization request for the SSSs is performed by the reduction's oracles. As, by assumption, the proof is wrong for σ_0^{SSS}, it breaks the signer-accountability of the underlying SSS in any case. Thus, $|\Pr[S_0] - \Pr[S_1]| \le \nu_{\mathsf{sss\text{-}sigacc}}(\kappa)$ follows.

Game 2: $(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \pi^{\mathsf{PS}}, m, \sigma^{\mathsf{PS}})$, where $\sigma^{\mathsf{PS}} = ((\sigma_i^{\mathsf{SSS}})_{0\le i\le \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1\le i\le \ell_m})$, where $(m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1\le i\le \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$ is not new, but σ_0^{SSS} has never been returned by the challenger.

Transition - Game 0 → Game 1: In this case, $(\mathsf{pk}_{\mathsf{sig}}^{\mathsf{SSS}}, \pi^{\mathsf{PS}}, (m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1\le i\le \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}}), \sigma_0^{\mathsf{SSS}})$ is a valid forgery of the outer SSS. The reduction works as in the prior hop. As, by assumption, the proof is wrong for σ_0^{SSS}, it breaks the signer-accountability of the underlying SSS in any case. Thus, $|\Pr[S_1] - \Pr[S_2]| \le \nu_{\mathsf{sss\text{-}sigacc}}(\kappa)$ follows.

Now, the adversary can no longer win the signer-accountability game. Moreover, each hop changes the view of the adversary only negligibly, concluding the proof.

Sanitizer-Accountability. To prove that our scheme is sanitizer-accountable, we use a sequence of games:

Game 0: The original sanitizer-accountability game.

Game 1: We now abort, if the adversary outputs $(m, \sigma^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$, where $\sigma^{\mathsf{PS}} = ((\sigma_i^{\mathsf{SSS}})_{0\le i\le \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1\le i\le \ell_m})$, where any $(m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1\le i\le \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$, protected by σ_0^{SSS}, has never been returned by the challenger.

Transition - Game 0 → Game 1: In this case, $((m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1\le i\le \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}}), \sigma_0^{\mathsf{SSS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{SSS}})$ is a valid forgery of the outer SSS. For the reduction, $\mathcal{B}$ receives the challenge keys $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{SSS}\prime}$ from its own challenger, and embeds them into $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}$. Every underlying signing and proof-generation request for the SSSs is performed by the reduction's oracles. As, by assumption, the signer outputs a wrong proof for σ_0^{SSS}, it breaks the sanitizer-accountability of the underlying SSS in any case. Thus, $|\Pr[S_0] - \Pr[S_1]| \le \nu_{\mathsf{sss\text{-}sanacc}}(\kappa)$ follows.

Game 2: We now abort, if the adversary outputs $(m, \sigma^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$, where $\sigma^{\mathsf{PS}} = ((\sigma_i^{\mathsf{SSS}})_{0\le i\le \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1\le i\le \ell_m})$, where any $(m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1\le i\le \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$ is not new, but σ_0^{SSS} has never been returned by the challenger.

Transition - Game 0 → Game 1: In this case, $((m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1\le i\le \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}}), \sigma_0^{\mathsf{SSS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{SSS}})$ is a valid forgery of the outer SSS. The reduction works as in the prior hop. As, by assumption, the signer outputs a wrong proof for σ_0^{SSS}, it breaks the sanitizer-accountability of the underlying SSS in any case. Thus, $|\Pr[S_1] - \Pr[S_2]| \le \nu_{\mathsf{sss\text{-}sanacc}}(\kappa)$ follows.

Now, the adversary can no longer win the sanitizer-accountability game. Moreover, each hop changes the view of the adversary only negligibly, concluding the proof.

Immutability. To prove that our scheme is immutable, we use a sequence of games:

Game 0: The original immutability game.
Game 1: We now abort, if the challenger draws a tag twice.
Transition - Game 0 → Game 1: The probability that this event happens is bounded by the birthday paradox. $|\Pr[S_0] - \Pr[S_1]| \leq q_t^2/2^\kappa$ follows, where q_t is the number of drawn tags.
Game 2: We now abort, if the adversary outputs $(m, \sigma^{\mathsf{PS}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$, where $\sigma^{\mathsf{PS}} = ((\sigma_i^{\mathsf{SSS}})_{0\leq i\leq \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1\leq i\leq \ell_m})$, but $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}$ was never signed by the signing oracle w.r.t. to τ.
Transition - Game 1 → Game 2: This breaks the immutability property of the outer SSS. The reduction proceeds as follows. It receives $\mathsf{pk}^{\mathsf{SSS}\prime}_{\mathsf{sig}}$ from its own challenger and embeds it into $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}$. Then, every signing query is performed by the reduction's own oracles. Then, after $\mathcal{A}$ returned $(m, \sigma^{\mathsf{PS}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$, $(m', \sigma_0^{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}})$ with $m' = (m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1\leq i\leq \ell_m}, \tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$ is a valid forgery. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{sss\text{-}imm}}(\kappa)$ follows.
Game 3: We now abort, if the adversary outputs $(m, \sigma^{\mathsf{PS}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$, where $\sigma^{\mathsf{PS}} = ((\sigma_i^{\mathsf{SSS}})_{0\leq i\leq \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1\leq i\leq \ell_m})$, but τ was never drawn by the challenger.
Transition - Game 2 → Game 3: As τ is non-admissible, the adversary was able to generate a signature not derivable, breaking the immutability of the outer SSS. The reduction works exactly as in the prior game. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{sss\text{-}imm}}(\kappa)$ follows.
Game 4: We now abort, if the adversary outputs $(m, \sigma^{\mathsf{PS}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$, where $\sigma^{\mathsf{PS}} = ((\sigma_i^{\mathsf{SSS}})_{0\leq i\leq \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1\leq i\leq \ell_m})$, but some τ_i was never signed by the challenger w.r.t. τ or the ordering is inconsistent.
Transition - Game 3 → Game 4: As each τ_i is signed by the RSS, the adversary was able to generate a forgery of the RSS. It receives $\mathsf{pk}^{\mathsf{RSS}\prime}_{\mathsf{sig}}$ from its own challenger and embeds it into $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}$. Then, every signing query is performed by the reduction's own oracles. Then, $((\tau_1, \tau_2, \ldots, \tau_\ell, \tau, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{sig}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}), \sigma^{\mathsf{RSS}})$ is a valid forgery. $|\Pr[S_4] - \Pr[S_5]| \leq \nu_{\mathsf{rss\text{-}unf}}(\kappa)$ follows.
Game 5: We now abort, if the adversary outputs $(m, \sigma^{\mathsf{PS}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$, where $\sigma^{\mathsf{PS}} = ((\sigma_i^{\mathsf{SSS}})_{0\leq i\leq \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1\leq i\leq \ell_m})$, but it was able to redact a block not marked as redactable.
Transition - Game 4 → Game 5: Note, we already ruled out tag-collisions and thus the messages are uniquely identifiable. The reduction is same as in the prior hop. $|\Pr[S_4] - \Pr[S_5]| \leq \nu_{\mathsf{rss\text{-}unf}}(\kappa)$ follows.
Game 6: We now abort, if the adversary outputs $(m, \sigma^{\mathsf{PS}}, \mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}})$, where $\sigma^{\mathsf{PS}} = ((\sigma_i^{\mathsf{SSS}})_{0\leq i\leq \ell_m}, \sigma^{\mathsf{RSS}}, c, \tau, (\tau_i)_{1\leq i\leq \ell_m})$, but it was able to sanitize a block with tag τ_i which was not marked as sanitizable.
Transition - Game 5 → Game 6: Note, we already ruled out tag-collisions and thus the messages are uniquely identifiable. The reduction is same as in Game 3, but the reduction returns $((m^i, \tau, \tau_i), \sigma_i^{\mathsf{SSS}}, \mathsf{pk}^{\mathsf{SSS}}_{\mathsf{san}})$. $|\Pr[S_5] - \Pr[S_6]| \leq \nu_{\mathsf{sss\text{-}imm}}(\kappa)$ follows.

Now, the adversary can no longer win the immutability game. Moreover, each hop changes the view of the adversary only negligibly, concluding the proof.

Privacy. To prove that our scheme is private, we use a sequence of games:

Game 0: The original privacy game, where $b = 0$.
Game 1: We now replace the encryption of $\mathsf{RED}^{\mathsf{RSS}}$ (and each $\mathsf{RED}^{\mathsf{RSS}\prime}$) with an encryption of zeroes with the same length if encrypted to pk_{Π} contained in $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}$.
Transition - Game 1 → Game 2: This does changes the view of the adversary only negligibly due to the IND-CCA2 security of Π. In particular, using a series of hybrids, replace each ciphertext one by one, while the challenge key $\mathsf{pk}_{\Pi}{}'$ is embedded into $\mathsf{pk}^{\mathsf{PS}}_{\mathsf{san}}$. The ciphertexts generated from any other signer can be decrypted using the decryption oracle provided, as the label is different due to the different public key, while they are known to the challenger in the other case without decrypting. So, if $\mathcal{A}$ notices a difference, so

does the reduction $\mathcal{B}$. Note, all oracles can still be simulated honestly, as the content the ciphertexts should contain is still known. $|\Pr[S_0] - \Pr[S_1]| \leq q_e \nu_{\mathsf{enc\text{-}ind\text{-}cca2}}$ follows, where q_e is the number of ciphertexts generated.

Game 2: Instead of signing (m_0^i, τ, τ_i) in the inner SSSs and adjusting them to (m^i, τ, τ_i), sign (m_1^i, τ, τ_i) and adjust it. Likewise, instead of signing $(m_0, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1 \leq i \leq \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$ in the outer SSS and then adjusting it, directly sign $(m_1, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1 \leq i \leq \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$ and adjust it to $(m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1 \leq i \leq \ell'_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$. Note, the distribution of c, σ^{RSS} and the tags are still exactly the same, even if reused. Moreover, the redactions are still performed as in the case $b = 0$.

Transition - Game 1 → Game 2: Assume that the adversary can distinguish these two games. We can then construct a reduction $\mathcal{B}$ which uses the adversary $\mathcal{A}$ to break the privacy of the underlying SSS. Namely, $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{SSS}\prime}$ and $\mathsf{pk}_{\mathsf{san}}^{\mathsf{SSS}\prime}$, and embeds them into $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}$ and $\mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}}$. Then, every signing, editing and proof oracle queries are answered by $\mathcal{B}$'s own oracles. However, for the calls to the $\mathsf{LoREdit}$ oracle, the calls for the SSSs are redirected to the LoRSan oracle and the result embedded to the answer. Clearly, the simulation is perfect. Then, whatever $\mathcal{A}$ outputs is also output by $\mathcal{B}$. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{sss\text{-}priv}}$ follows.

Game 3: Instead of signing $(\tau_1, \tau_2, \ldots, \tau_\ell, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$ in the RSSs from the first message, use the second message and then redact as required. Note, the distribution of the tags are still exactly the same due to the uniform distribution.

Transition - Game 2 → Game 3: Assume that the adversary can distinguish these two games. We can then construct a reduction $\mathcal{B}$ which uses the adversary $\mathcal{A}$ to break the privacy of the underlying RSS. Namely, $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}\prime}$ and embeds them into $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}$. Then, every signing query is answered by $\mathcal{B}$'s own signing oracle. However, for the calls to the $\mathsf{LoREdit}$ oracle, the calls for the RSSs are redirected to the $\mathsf{LoRRedact}$ oracle and the result embedded to the answer. Clearly, the simulation is perfect. Then, whatever $\mathcal{A}$ outputs is also output by $\mathcal{B}$. Note, here we no longer need $\mathsf{RED}^{\mathsf{RSS}}$, as this done via the oracles. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{rss\text{-}priv}}$ follows.

Now, we are in the case $b = 1$. As each hop only changes the view of the adversary negligibly, privacy is proven.

Transparency. To prove that our scheme is transparent, we use a sequence of games:

Game 0: The original transparency game, where $b = 0$.

Game 1: We now replace the encryption of $\mathsf{RED}^{\mathsf{RSS}}$ (and each $\mathsf{RED}^{\mathsf{RSS}\prime}$) with an encryption of zeroes with the same length if encrypted to pk_{Π} contained in $\mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}}$.

Transition - Game 1 → Game 2: This does changes the view of the adversary only negligibly due to the IND-CCA2 security of Π. In particular, using a series of hybrids, replace each ciphertext one by one, while the challenge key $\mathsf{pk}_{\Pi}{}'$ is embedded into $\mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}}$. The ciphertexts generated from any other signer can be decrypted using the decryption oracle provided, as the label is different due to the different public key, while they are known to the challenger in the other case without decrypting. So, if $\mathcal{A}$ notices a difference, so does the reduction $\mathcal{B}$. Note, all oracles can still be simulated honestly, as the content the ciphertexts should contain is still known. $|\Pr[S_0] - \Pr[S_1]| \leq q_e \nu_{\mathsf{enc\text{-}ind\text{-}cca2}}$ follows, where q_e is the number of ciphertexts generated.

Game 2: Instead of signing (m, τ, τ_i) in the inner SSSs and adjusting them to (m', τ, τ_i), directly sign (m', τ, τ_i). Likewise, instead of signing $(m, \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1 \leq i \leq \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$ in the outer SSS and then adjusting it, directly sign $(m', \sigma^{\mathsf{RSS}}, c, (\tau_i, \sigma_i^{\mathsf{SSS}})_{1 \leq i \leq \ell_m}, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$. Again, the distribution of c, σ^{RSS} and the tags are still exactly the same, even if reused. Moreover, the redactions are still performed as in the case $b = 0$. Note, the restrictions on the proof-oracle are still implicitly enforced.

Transition - Game 1 → Game 2: Assume that the adversary can distinguish these two games. We can then construct a reduction $\mathcal{B}$ which uses the adversary $\mathcal{A}$ to break the transparency of the underlying SSS.

Namely, $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{SSS}\prime}$ and $\mathsf{pk}_{\mathsf{san}}^{\mathsf{SSS}\prime}$, and embeds them into $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}$ and $\mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}}$. Then, every signing, editing and proof oracle queries are answered by $\mathcal{B}$'s own oracles. However, for the calls to the $\mathsf{Sign}/\mathsf{Edit}$ oracle, the calls for the SSSs are redirected to the $\mathsf{Sign}/\mathsf{Sanit}$ oracle and the result embedded to the answer. Clearly, the simulation is perfect. Then, whatever $\mathcal{A}$ outputs is also output by $\mathcal{B}$. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{sss\text{-}tran}}$ follows.

Game 3: Instead of signing $(\tau_1, \tau_2, \ldots, \tau_\ell, \tau, \mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}, \mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}})$ in the RSSs from the first message and then redacting it, directly sign the redacted messages. Note, the distribution of the tags are still exactly the same due to the uniform distribution.

Transition - Game 2 → Game 3: Assume that the adversary can distinguish these two games. We can then construct a reduction $\mathcal{B}$ which uses the adversary $\mathcal{A}$ to break the transparency of the underlying RSS. Namely, $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{RSS}\prime}$ and embeds them into $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}$. Then, every signing query is answered by $\mathcal{B}$'s own signing oracle. However, for the calls to the $\mathsf{Sign}/\mathsf{Edit}$ oracle, the calls for the RSSs are redirected to the $\mathsf{Sign}/\mathsf{Redact}$ oracle and the result embedded to the answer. Clearly, the simulation is perfect. Then, whatever $\mathcal{A}$ outputs is also output by $\mathcal{B}$. Note, here we no longer need $\mathsf{RED}^{\mathsf{RSS}}$, as this done via the oracles and are already replaced with a 0. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{rss\text{-}tran}}$ follows.

Now, we are in the case $b = 1$. As each hop only changes the view of the adversary negligibly, transparency is proven.

Invisibility. To prove that our scheme is invisible, we use a sequence of games:

Game 0: The original invisibility game, where $b = 0$.

Game 1: Instead of using $\mathsf{ADM}_0^{\mathsf{PS}}.1$ use $\mathsf{ADM}_1^{\mathsf{PS}}.1$ as $\mathsf{ADM}^{\mathsf{SSS}}$ in the SSS.

Transition - Game 0 → Game 1: This does changes the view of the adversary only negligibly due to the invisibility of the underlying SSS. Namely, assume that an adversary $\mathcal{A}$ can distinguish these games with non-negligible probability. We can then construct an adversary $\mathcal{B}$ which breaks the invisibility guarantees of the used SSS. In particular, $\mathcal{B}$ receives $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{SSS}\prime}$ and $\mathsf{pk}_{\mathsf{san}}^{\mathsf{SSS}\prime}$, and embeds them into $\mathsf{pk}_{\mathsf{sig}}^{\mathsf{PS}}$ and $\mathsf{pk}_{\mathsf{san}}^{\mathsf{PS}}$. For all oracle queries, $\mathcal{B}$ uses its own oracles to answer correctly, but makes block 1 in each underlying SSS admissible or not using its own challenge oracle. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{sss\text{-}invis}}$ follows.

Now, we are in the case $b = 1$. As each hop only changes the view of the adversary negligibly, invisibility is proven.