Redactable vs. Sanitizable Signatures

Kai Samelin, Henrich C. Pöhls,
Joachim Posegga and Hermann de Meer
{ks, hp, jp}@sec.uni-passau.de, demeer@uni-passau.de

Institute of IT-Security and Security-Law (ISL), University of Passau, Germany

Technical Report, Number MIP-1208
Department of Informatics and Mathematics
University of Passau, Germany
December 2012
Redactable vs. Sanitizable Signatures

Kai Samelin\textsuperscript{1}, Henrich C. Pöhls\textsuperscript{2}, Joachim Posegga\textsuperscript{2}, Hermann de Meer\textsuperscript{1}

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

Abstract. Malleable signature schemes allow altering signed data in a controlled way while keeping the signature verifiable trusting the signer’s key. Several constructions exist. They can be grouped in two different categories: (1) redactable signatures (RSS) and (2) sanitizable signatures (SSS). RSSs allow for removing blocks of a signed document, while SSSs offer the possibility to change all admissible blocks to arbitrary strings. This paper shows that SSSs with a strengthened security definition can be transformed into RSSs with a weakened privacy definition. A transformation from a RSS into a SSS is not possible, even if we assume accountability for RSS. In particular, no unforgeable RSS can be transformed into a SSS. This work provides the first rigorous proof that RSSs and SSSs are two different concepts.

Keywords: Redactable Signatures, Sanitizable Signatures, Privacy, Malleable Signatures

1 Introduction

Standard digital signature schemes like RSA-PSS \cite{5, 35} become invalid on any change to the signed data protected. However, this also prohibits a third party from changing this data in a controlled way. Applications, where such a controlled is crucial, include secure routing \cite{2} or the anonymization of medical data \cite{24}. It is also of paramount importance that the modification of the signed data requires no interaction between the sanitizer, i.e., the party who changes the signed data, and the original signer. This addresses constellations where the original signer is not reachable anymore, e.g., in case of death, or in the case where the signer may be reachable, but must not know which data is passed to other third parties. This may happen, if personal data like gender, age or place of birth is involved. A suitable approach to this so-called “digital document

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

\textsuperscript{**}Is funded by BMBF (FKZ:13N10966) and ANR as part of the ReSCUeIT project
sanitization problem” [30] are malleable signature schemes. Malleable signature schemes allow for certain controlled changes to signed data such that the resulting changed message’s authenticity is still verifiable. Let $m = (m[1], \ldots, m[\ell])$, where $\ell \in \mathbb{N}^*$, be a string $m$ split up into $\ell$ parts we refer to as blocks. A redactable signature scheme ($RSS$) allows everyone to remove blocks $m[i]$ from $m$. In particular, a redaction of the block $m[i]$ leaves a blinded message $m'$ without $m[i]$, i.e., $m' = (\ldots, m[i-1], \bot, m[i+1], \ldots)$. If $\bot$ is visible has a major impact on the security model of the $RSS$. It also requires that the third party can derive a signature $\sigma'$ which verifies for $m'$. On the other hand, a sanitizable signature scheme ($SSS$) allows that a sanitizer, which has its own secret key, can change the admissible blocks, defined by the signer, into arbitrary strings $m[i]' \in \{0, 1\}^*$. Hence, the sanitizer can generate a verifiable message/signature pair $(m', \sigma')$, where $\sigma'$ is the corresponding signature and $m' = (\ldots, m[i]', \ldots)$. Obviously, the derived signatures still need to induce trust, i.e., it must be verifiable that all changes were endorsed by the signer. On the first sight, both approaches aim for the same goal, i.e., sanitizing signed data. However, $SSS$s only allow the alteration of blocks, while $RSS$s only allow the removal of complete blocks. Moreover, $SSS$s require an additional key pair for sanitization, while $RSS$s allow for public redactions, i.e., no additional key pair is needed.

**Motivation.** Sanitizing digitally signed data becomes more and more important as digital signatures are deployed in business settings and for timestamping or archiving purposes [33, 39]. However, standard unforgeable digital signature do not allow any later modification to the data signed. Hence, we need to find solutions to the “digital document sanitization problem” [30]. Current provably secure solutions only focus on one specific type of malleable signature scheme, i.e., $RSS$s or $SSS$s, even though they see them as related work. Hence, the question arises how the relation between both types of malleable signature is. This paper addresses this gap and proves the minimal set of security definitions required to transform a $SSS$ into a $RSS$. We also prove that no transform can result in a fully private $RSS$. This shows, that every existing transform is not secure, as the security models are not sufficient. Hence, we conclude that $RSS$s and $SSS$s must be combined to achieve a maximum amount of flexibility and security.

### 1.1 Contribution

This paper rigorously shows that $RSS$s and $SSS$s are completely different concepts, with one notable exception: weakening the privacy definition of $RSS$s, while altering the security definitions of $SSS$s, a $SSS$ can be transformed into a $RSS$. We provide a general algorithm for the transform and show that an $SSS$ that is
– strongly private,
– weakly immutable and
– weakly blockwise non-interactive publicly accountable

can be transformed into a weakly private \textit{RSS}. We prove that this is the minimal set of assumptions required. \textit{Strongly private} for SSSs requires, that even if the sanitizing key is known, no statements about the original message can be made. \textit{Weak blockwise non-interactive public accountability} prohibits a sanitizer from accusing the signer for a specific block. A \textit{weakly immutable} SSS prohibits an adversary knowing the secret sanitizing key from altering blocks not designated to be sanitized. However, we also prove that the resulting RSSs are only weakly private. In a \textit{weakly private} RSS, a third party can see where the message has been redacted, i.e., it sees the position of the change, but cannot make any additional statements, i.e., cannot revert the redaction.

We introduce the required formal security model in Sect. 2. Moreover, we show that no unforgeable RSS can be transformed into a SSS. This results in the fact that RSSs and SSSs must be combined to achieve more flexibility in sanitizing signed data. Hence, our results rule out every existing transformations, as the existing security models are not suitable. We consider this the main contribution of this paper.

\textbf{State-of-the-Art.} SSSs have been introduced by Ateniese et al. [2] at ESORICS’05. Brzuska et al. later formalized the most used security properties [7]. These have been later extended for unlinkability [9] and (blockwise) non-interactive public accountability [10]. Moreover, several extensions like limiting-to-values [11, 22, 31], trapdoor SSSs [13, 41] and multi-sanitizer environments [8, 12] have been considered. Currently, the only work considering SSSs and data-structures more complex than lists appeared in [31].

On the other hand, RSSs have been introduced in [21], and in a slightly different way in [38]. Based on their work, many additional research appeared. Several cover more complex data-structures like trees [6, 24, 34, 31, 36] and graphs [26]. The standard security properties of RSSs have been formalized in [6, 14, 32, 36]. How to force the signer to commit to a given message has been shown in [33], while Ahn et al. introduce the notion of statistically unlinkable RSSs [1]. Even stronger privacy notions have been introduced in [3]. However, the scheme by Ahn et al. only achieves the less common notion of selective unforgeability [1]. There exists many additional work on RSSs. However, most of the schemes are not private, e.g., [18–20, 27, 29, 40]. Hence, a verifier can make statements about the original message \(m\), which contradicts the intention of a RSS [6].

Combinations of both approaches appeared in [18–20]. However, as already pointed out by Samelin et al., their schemes do not preserve privacy [37]. No other work considering combinations or relations of SSSs and RSSs is known to the authors.
1.2 Outline

The rest of the paper is structured as follows. In Sect. 2, we give the required preliminaries to understand our results. This section also introduces the new notions of strong privacy, weak immutability and weak blockwise non-interactive public accountability for SSSs. It also provides the new definition of weak privacy for RSSs. A general transform showing that a SSS with strong privacy, weak immutability and weak blockwise non-interactive public accountability can be transformed into an RSS with weak privacy is given in Sect. 3. Based on the preliminaries, we give formal proofs of the relation between SSSs and RSSs in Sect. 4. Our resulting scheme is subject to further modifications in Sect. 5. We conclude our work in Sect. 6. Additional formal proofs of security are found in the appendix.

2 Preliminaries

For a message \(m = (m[1], \ldots, m[\ell])\), we call \(m[i] \in \{0,1\}^*\) a block, while \("\) denotes a uniquely reversible concatenation of blocks or strings. The symbol \(\bot \notin \{0,1\}^*\) denotes an error or an exception. For a visible redaction, we use the symbol \(\square \notin \{0,1\}^*, \square \neq \bot\).

2.1 Sanitizable Signatures

The used nomenclature is adapted from Brzuska et al. [7], which is also true for the following definition:

Definition 1 (Sanitizable Signature Scheme). Any SSS consists of at least seven efficient, i.e., PPT algorithms. In particular, let \(SSS := (KGen_{sig}, KGen_{san}, Sign, Sanit, Verify, Proof, Judge)\), such that:

Key Generation. There are two key generation algorithms, one for the signer and one for the sanitizer. Both create a pair of keys, a private key and the corresponding public key, based on the security parameter \(\lambda\):

\[
(pk_{sig}, sk_{sig}) \leftarrow KGen_{sig}(1^\lambda) \\
(pk_{san}, sk_{san}) \leftarrow KGen_{san}(1^\lambda)
\]

Signing. The Sign algorithm takes as input the security parameter \(\lambda\), a message \(m = (m[1], \ldots, m[\ell])\), \(m[i] \in \{0,1\}^*\), the secret key \(sk_{sig}\) of the signer, the public key \(pk_{san}\) of the sanitizer, as well as a description ADM of the admissibly modifiable blocks, where ADM contains the number \(\ell\) of blocks in \(m\), as well the indices of the modifiable blocks. It outputs the message \(m\) and a signature \(\sigma\) (or \(\bot\), indicating an error):

\[
(m, \sigma) \leftarrow Sign(1^\lambda, m, sk_{sig}, pk_{san}, ADM)
\]
Sanitizing. Algorithm Sanit takes a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$, the security parameter $\lambda$, a signature $\sigma$, the public key $pk_{sig}$ of the signer and the secret key $sk_{san}$ of the sanitizer. It modifies the message $m$ according to the modification instruction $\text{MOD}$, which contains pairs $(i, m[i'])$ for those blocks that shall be modified. Sanit calculates a new signature $\sigma'$ for the modified message $m' \leftarrow \text{MOD}(m)$. Then Sanit outputs $m'$ and $\sigma'$ (or possibly $\perp$ in case of an error).

$$(m', \sigma') \leftarrow \text{Sanit}(1^\lambda, m, \text{MOD}, \sigma, pk_{sig}, sk_{san})$$

Verification. The Verify algorithm outputs a decision $d \in \{\text{true, false}\}$ verifying the correctness of a signature $\sigma$ for a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$ with respect to the public keys $pk_{sig}$ and $pk_{san}$ and the security parameter $\lambda$:

$$d \leftarrow \text{Verify}(1^\lambda, m, \sigma, pk_{sig}, pk_{san})$$

Proof. The Proof algorithm takes as input the security parameter, the secret signing key $sk_{sig}$, a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$ and a signature $\sigma$ as well a set of (polynomially many) additional message/signature pairs $\{(m_i, \sigma_i) \mid i \in \mathbb{N}^+\}$ and the public key $pk_{san}$. It outputs a string $\pi \in \{0,1\}^*$ (or $\perp$, indicating an error):

$$\pi \leftarrow \text{Proof}(1^\lambda, sk_{sig}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}^+\}, pk_{san})$$

Judge. Algorithm Judge takes as input the security parameter, a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$ and a valid signature $\sigma$, the public keys of the parties and a proof $\pi$. It outputs a decision $d \in \{\text{Sig, San, } \perp\}$ indicating whether the message/signature pair has been created by the signer or the sanitizer (or $\perp$, indicating an error):

$$d \leftarrow \text{Judge}(1^\lambda, m, \sigma, pk_{sig}, pk_{san}, \pi)$$

To have an algorithm actually able to derive the accountable party for a specific block $m[i]$, Brzuska et al. introduced the additional algorithm Detect [10]. The algorithm Detect is not part of the original SSS description by Ateniese et al., since it is not required for the purpose of a SSS [2, 7]. However, it is required to later define (weak) blockwise non-interactive public accountability (See Def. 6).

Definition 2 ((SSS) Detect). On input of the security parameter $\lambda$, a message/signature pair $(m, \sigma)$, the corresponding public keys $pk_{sig}$ and $pk_{san}$, and the block index $1 \leq i \leq \ell$, Detect outputs the accountable party (San or Sig) for block $i$ (or $\perp$, indicating an error):

$$d \leftarrow \text{Detect}(1^\lambda, m, \sigma, pk_{sig}, pk_{san}, i), d \in \{\text{San, Sig, } \perp\}$$
We require the usual correctness properties to hold. In particular, all genuinely signed or sanitized messages are accepted, while every genuinely created proof by the signer leads the judge to decide in favor of the signer. For a formal definition of correctness, refer to [7, 10]. It is also required by every SSS that ADM is always correctly recoverable from any valid message/signature pair \((m, \sigma)\).

Jumping ahead, we want to emphasize that a SSS with weak non-interactive public accountability has an empty Proof algorithm and a Judge that detects any sanitization, based on any proof \(\pi \in \{0, 1\}^* \cup \perp\). We give formal definitions of the security properties in a game-based manner after introducing RSSs.

### 2.2 Redactable Signatures

The following notation is derived from [6] and [37].

**Definition 3 (Redactable Signature Schemes).** A RSS consists of four efficient algorithms. In particular, let \(\mathcal{RSS} := (\text{KeyGen}, \text{Sign}, \text{Verify}, \text{Redact})\) such that:

**KeyGen.** The algorithm KeyGen outputs the public key and private key of the signer, where \(\lambda\) is the security parameter:

\[(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)\]

**Sign.** The algorithm Sign gets as input the security parameter \(\lambda\), the secret key \(sk\) and the message \(m = (m[1], \ldots, m[\ell]), m[i] \in \{0, 1\}^*:\)

\[(m, \sigma) \leftarrow \text{Sign}(1^\lambda, sk, m)\]

**Verify.** The algorithm Verify outputs a decision \(d \in \{\text{true, false}\},\) indicating the correctness of the signature \(\sigma\), w.r.t. \(pk\), protecting \(m = (m[1], \ldots, m[\ell]), m[i] \in \{0, 1\}^*:\)

\[d \leftarrow \text{Verify}(1^\lambda, pk, m, \sigma)\]

**Redact.** The algorithm Redact takes as input the message \(m = (m[1], \ldots, m[\ell]), m[i] \in \{0, 1\}^*\), the public key \(pk\) of the signer, a valid signature \(\sigma\), a list of indices \(\text{MOD}\) of blocks to be redacted and the security parameter \(\lambda\). It returns a modified message \(m' \leftarrow \text{MOD}(m)\) (or \(\perp\), indicating an error):

\[(m', \sigma') \leftarrow \text{Redact}(1^\lambda, pk, m, \sigma, \text{MOD})\]

We denote the transitive closure of \(m\) as \(\text{span}_w(m)\). This set contains all messages derivable from \(m\) w.r.t. Redact.

As for SSSs, the correctness properties for RSSs are required to hold as well. Thus, every genuinely signed or redacted message must verify. Refer to [6] for a formal definition of correctness.
Security Models. In this section, the needed security properties and models required for our proofs are introduced. They are derived from [7, 17, 37]. The requirement that ADM is always correctly reconstructable is captured within the unforgeability and immutability definitions. Note, following [7], a SSS must at least be unforgeable, immutable, accountable and private to be meaningful. Hence, we assume that all used SSSs fulfill these four fundamental security requirements; if these requirements are not met, the construction is not considered a SSS and the results of this paper may differ. For the following definitions of security properties, we merge descriptions of SSSs and RSSs where possible, allowing to see the parallels.

Definition 4 ((RSS) Unforgeability). No one should be able to compute a valid signature on a message not previously queried without having access to any private keys [6]. That is, even if an outsider can request signatures on different documents, it remains hard to forge a signature for a document not previously signed. This is analogous to the standard unforgeability requirement for standard signature schemes [16], except that it excludes valid redactions from the set of forgeries. We say that a RSS is unforgeable, if for any efficient (PPT) adversary \( A \) the probability that the game depicted in Fig. 1 returns 1, is negligible (as a function of \( \lambda \)).

Experiment Unforgeability\(^{RSS}(\lambda)\)
\[
(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)
(m^*, \sigma^*) \leftarrow A_{\text{Sign}(sk)}(pk)
\]
let \( i = 1, \ldots, q \) index the queries
return 1, if
\[
\text{Verify}(pk, m^*, \sigma^*) = 1 \text{ and } \\
\forall i, 1 \leq i \leq q : m^* \not\in \text{span}_m(m_i)
\]

Fig. 1. Unforgeability for RSSs

Definition 5 ((SSS) Unforgeability). As before, no one should be able to generate valid signatures on new documents not queried before without having access to any private keys. For SSSs, we also have to take the sanitization and proof oracles into account [7]. Again, this is analogous to the standard unforgeability requirement for standard signature schemes [16], except that it excludes valid sanitizations from the set of forgeries. We say that a SSS is unforgeable, if for any efficient (PPT) adversary \( A \) the probability that the game depicted in Fig. 2 returns 1, is negligible (as a function of \( \lambda \)).

Definition 6 ((SSS) Weak Blockwise Non-Interactive Public Accountability). A sanitizable signature scheme SSS is weakly non-interactive publicly
Experiment Unforgeability\textsubscript{\text{SSS}}(\lambda)

\begin{align*}
(p_{\text{sig}}, s_{\text{sig}}) &\leftarrow \text{KGen}_{\text{sig}}(1^\lambda) \\
(p_{\text{san}}, s_{\text{san}}) &\leftarrow \text{KGen}_{\text{san}}(1^\lambda) \\
(m^*, \sigma^*) &\leftarrow \mathcal{A}_{\text{Sanit}}(\cdot, s_{\text{san}}) (p_{\text{sig}}, p_{\text{san}}) \\
\text{return } 1, \text{ if } & \\
\text{Verify}(m^*, \sigma^*, p_{\text{sig}}, p_{\text{san}}) = \text{true} \text{ and } & \\
(\text{ADM}_i \neq \text{ADM}^* \text{ or } & \\
m^* \text{ has not been returned by an oracle}) \\
\end{align*}

Fig. 2. Unforgeability for \text{SSS}s

Experiment WBlockPubAcc\textsubscript{\text{SSS}}(\lambda)

\begin{align*}
(p_{\text{sig}}, s_{\text{sig}}) &\leftarrow \text{KGen}_{\text{sig}}(1^\lambda) \\
(p_{\text{san}}, s_{\text{san}}) &\leftarrow \text{KGen}_{\text{san}}(1^\lambda) \\
(p^*, m^*, \sigma^*) &\leftarrow \mathcal{A}_{\text{Sanit}}(\cdot, s_{\text{san}}, p_{\text{san}}) (p_{\text{sig}}, s_{\text{san}}, p_{\text{sig}}) \\
\text{Let } (m_i, \text{ADM}_i, p_{\text{san}}) \text{ and } \sigma_i \text{ for } i = 1, \ldots, k & \\
\text{be the queries/answers to/from } \mathcal{O}_{\text{Sig}} & \\
\text{return } 1, \text{ if } & \\
\text{Verify}(1^\lambda, m^*, \sigma^*, p_{\text{sig}}, p^*) = \text{true}, \text{ and } & \\
\text{for all } m_i \text{ with } p_{\text{san}, i} = p^*, \exists q, \text{ s.t. } & \\
\text{Detect}(1^\lambda, m^*, \sigma^*, p_{\text{sig}}, p^*, q) = \text{Sig} & \\
\text{and } (q, m_i[q]) \in \text{MOD}_i. & \\
\text{return } 0 & \\
\end{align*}

Fig. 3. Weak Blockwise Non-Interactive Public Accountability for \text{SSS}s

accountable, if \text{Proof} = \bot, and if for any efficient algorithm \mathcal{A} the probability that the experiment given in Fig. 3 returns 1 is negligible (as a function of \lambda).

The basic idea is that an adversary, i.e., the sanitizer, has to be able to make the \text{Detect} algorithm accuse the signer, if it did not sign the specific block. Please note, the sanitizer key is not generated by the adversary. An example for a weakly blockwise non-interactive publicly accountable \text{SSS} is the scheme introduced by Brzuska et al. [10]. Note, in our definition the signer is not considered adversarial, contrary to Brzuska et al. [10]. Hence, we do not need to consider the case where the signer accuses the sanitizer, as done in [10]. We explain the reasons for our adversary model after the introduction of all required security properties.

Definition 7 ((\text{SSS}) Standard Privacy). No one should be able to gain any knowledge about sanitized parts without having access to them [7]. This is similar to the standard indistinguishability notion for encryption schemes [15]. The basic idea is that the oracle either signs and sanitizes the first message or the second, while the resulting message must be the same for each input. The adversary must not be able to decide which input message was used. We say that a \text{SSS} is
strongly private, if for any efficient (PPT) adversary \( A \) the probability that the game depicted in Fig. 4 returns 1, is negligibly close to \( \frac{1}{2} \) (as a function of \( \lambda \)).

\[ \textbf{Experiment \: Privacy}_{A}^{\text{SSS}}(\lambda) \]
\[
(\pk_{\text{sig}}, \sk_{\text{sig}}) \leftarrow \text{KGen}_{\text{sig}}(1^{\lambda}) \\
(\pk_{\text{san}}, \sk_{\text{san}}) \leftarrow \text{KGen}_{\text{san}}(1^{\lambda}) \\
b \leftarrow \{0, 1\} \\
a \leftarrow A_{\text{LoRSanit}}(\cdots, \sk_{\text{sig}}, \sk_{\text{san}}, b, \text{Sanit}(\cdots, \sk_{\text{san}})(\pk_{\text{sig}}, \pk_{\text{san}})) \\
\text{where oracle } \text{LoRSanit} \text{ on input of:} \\
m_0, \mod_0, m_1, \mod_1, \text{ADM} \\
\text{if } \mod_0(m_0) \neq \mod_1(m_1), \text{ return } \bot \\
\text{let } (m, \sigma) \leftarrow \text{Sign}(m_0, \sk_{\text{sig}}, \pk_{\text{san}}, \text{ADM}) \\
\text{return } (m', \sigma') \leftarrow \text{Sanit}(m, \mod_0, \sigma, \pk_{\text{sig}}, \sk_{\text{san}}) \\
\text{return } 1, \text{ if } a = b \\
\]

Fig. 4. Standard Privacy for SSSs

\textbf{Definition 8 ((SSS) Strong Privacy).} The basic idea remains the same: no one should be able to gain any knowledge about sanitized parts without having access to them, with one exception: the adversary is given the secret key \( \sk_{\text{san}} \) of the sanitizer. Hence, the adversary must not be able to decide which input message was used. We say that a SSS for documents is private, if for any efficient (PPT) adversary \( A \) the probability that the game depicted in Fig. 5 returns 1, is negligibly close to \( \frac{1}{2} \) (as a function of \( \lambda \)). This notion extends the definition of standard privacy [7] to also account for parties knowing the secret sanitizer key \( \sk_{\text{san}} \). Examples for strongly private SSSs are the scheme introduced by Brzuska et al. [10], as both of their schemes are information-theoretically private.

The privacy definition in [7] only considers outsiders as adversarial. However, we require that even insiders, i.e., sanitizers, are not able to win the game. This is similar to the game given in [12], with the notable exception that the key \( \sk_{\text{san}} \) is not generated by the adversary, only known to it. We explain the need for this alteration after the next definitions.

\textbf{Definition 9 ((RSS) Weak Privacy).} In a weakly private RSS, a third party can derive which parts of a message have been redacted without gathering more information, as redacted blocks are replaced with \( \bot \). The basic idea is that the oracle either signs and sanitizes the first message or the second. As before, the resulting redacted message \( m' \) must be the same for both inputs, with one additional exception: the length of both inputs must be the same, while \( \bot \) is considered part of the message. For strong privacy, this constraint is not required. We say that
**Experiment SPrivacy** \(\mathcal{S}\) (**SSS**) \((\lambda)\)

\[
\begin{align*}
(pk_{\text{sig}}, sk_{\text{sig}}) & \leftarrow \text{KGen}_{\text{sig}}(1^\lambda) \\
(pk_{\text{san}}, sk_{\text{san}}) & \leftarrow \text{KGen}_{\text{san}}(1^\lambda) \\
b & \leftarrow \{0, 1\} \\
a & \leftarrow \mathcal{A}^{\text{Sign}(sk_{\text{sig}}, \cdots), \text{Proof}(sk_{\text{sig}}, \cdots)}_{\text{LoRSanit}(\cdots, sk_{\text{san}}, sk_{\text{sig}}, b)}(pk_{\text{sig}}, pk_{\text{san}}, sk_{\text{san}})
\end{align*}
\]

where oracle LoRSanit on input of:

- \(m_0, \text{MOD}_0, m_1, \text{MOD}_1, \text{ADM}\)
- if \(\text{MOD}_0(m_0) \neq \text{MOD}_1(m_1)\), return \(\bot\)

let \((m, \sigma) \leftarrow \text{Sign}(m_0, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})\)

return \((m', \sigma') \leftarrow \text{Sanit}(m, \text{MOD}_0, \sigma, pk_{\text{sig}}, sk_{\text{san}})\)

return 1, if \(a = b\)

---

**Fig. 5. Strong Privacy for SSSs**

**Experiment WPrivacy** \(\mathcal{W}\) (**RSS**) \((\lambda)\)

\[
\begin{align*}
(pk, sk) & \leftarrow \text{KeyGen}_A(1^\lambda) \\
b & \leftarrow \{0, 1\} \\
d & \leftarrow \mathcal{A}^{\text{Sign}(sk, \cdots)}_{\text{LoRRedact}(\cdots, b)}(pk)
\end{align*}
\]

where oracle LoRRedact

for input \(m_0, m_1, \text{MOD}_0, \text{MOD}_1\):

- if \(\text{MOD}_0(m_0) \neq \text{MOD}_1(m_1)\), return \(\bot\)

Note, **visible** redacted parts are denoted \(\Box\), which are considered part of the message

\[
\begin{align*}
(m, \sigma) & \leftarrow \text{Sign}(sk, m_0) \\
return (m', \sigma') & \leftarrow \text{Redact}(pk, m, \sigma, \text{MOD}_0).
\end{align*}
\]

return 1, if \(b = d\)

---

**Fig. 6. Weak Privacy for RSSs**

A **RSS** for documents is **weakly** private, if for any efficient (PPT) adversary \(A\) the probability that the game depicted in Fig. 6 returns 1, is negligibly close to \(\frac{1}{2}\) (as a function of \(\lambda\)). We want to emphasize, that Lim et al. define weak privacy in a different manner, i.e., they prohibit access to the signing oracle [27].

Our definition allows such adaptive queries. Summarized, weak privacy just makes statements about blocks, not the complete message. See [24] for possible attacks. Weakly private schemes, following our definition, are, e.g., [18–20, 24, 25, 27].

In their schemes, the adversary is able to pinpoint the indices of the redacted blocks, as \(\bot\) is visible.

**Definition 10 ((RSS) Strong Privacy).** This definition is similar to weak privacy. However, redacted parts are not considered part of the message. We say that a **RSS** for documents is **strongly** private, if for any efficient (PPT) adversary \(A\) the probability that the game depicted in Fig. 7 returns 1, is negligibly close to \(\frac{1}{2}\) (as a function of \(\lambda\)). This is the standard definition of privacy [6, 37].
**Experiment SPrivacy**

\[(pk, sk) \leftarrow \text{KeyGen}(\lambda)\]

\[b \leftarrow \{0, 1\}\]

\[d \leftarrow A^{\text{LoRRedact}(\ldots, sk, b)}(pk)\]

where oracle LoRRedact

for input \(m_0, m_1, \text{MOD}_0, \text{MOD}_1:\)

if \(\text{MOD}_0(m_0) \neq \text{MOD}_1(m_1),\) return \(\bot\)

redacted blocks are not considered part of the message

\((m, \sigma) \leftarrow \text{Sign}(sk, m)\)

return \((m', \sigma') \leftarrow \text{Redact}(pk, m, \sigma, \text{MOD}_0)\).

return 1, if \(b = d\)

**Fig. 7. Strong Privacy for RSSs**

**Definition 11 ((RSS) Transparency).** Another well-known security definition is transparency [6, 32]. Interpreting the formal definition, which is depicted in Fig. 8, transparency is the anonymity of the signer, i.e., a third party cannot decide whether a given message/signature pair \((m, \sigma)\) originates from the signer or the sanitizer. A redactable signature scheme RSS is transparent, if for any efficient algorithm \(A\) the probability that the experiment given in Fig. 8 returns 1 is negligibly close to \(\frac{1}{2}\) (as a function of \(\lambda\)). The basic idea is that an adversary has access to an oracle which either signs and then sanitizes the message or vice versa. Following the argumentation and proofs given in [10], we can derive that no weakly blockwise non-interactive publicly accountable scheme can be transparent [10].

**Experiment Transparency**

\[(pk, sk) \leftarrow \text{KeyGen}(\lambda)\]

\[b \leftarrow \{0, 1\}\]

\[d \leftarrow A^{\text{Sign}(sk, \cdot), \text{Sign/Redact}(\ldots, sk, b)}(pk)\]

where oracle Sign/Redact for input \(m, \text{MOD}:\)

if \(\text{MOD}(m) \notin \text{span}_b(m),\) return \(\bot\)

if \(b = 0: (m, \sigma) \leftarrow \text{Sign}(sk, m),\)

\((m', \sigma') \leftarrow \text{Redact}(pk, \sigma, m, \text{MOD})\)

if \(b = 1: m' \leftarrow \text{MOD}(m)\)

\((m', \sigma') \leftarrow \text{Sign}(sk, m')\),

finally return \((m', \sigma')\).

return 1, if \(b = d\)

**Fig. 8. Transparency for RSSs**
Experiment Immutability\textsuperscript{SSS}(\lambda)

\((pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \text{KeyGen}(1^\lambda)\)

\((m^*, \sigma^*, pk_{\text{san}}^*) \leftarrow \mathcal{A}^{\text{Sign}(\cdot, sk_{\text{sig}}, \cdot), \text{Proof}(sk_{\text{sig}}, \cdot)}(pk_{\text{sig}})\)

return 1, if:

\(\text{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk_{\text{san}}^*) = \text{true}\)

\(\exists i: \, m^*[j_i] \neq m_i[j_i]\) for some \(j_i \notin \text{ADM}_i\), or

\(pk_{\text{san}}^* \neq pk_{\text{san}}, i\)

shorter messages are padded with \(\bot\) or \(\text{ADM}_i \neq \text{ADM}^*\)

Fig. 9. Immutability for \textit{SSS}s

Definition 12 ((\textit{SSS}) Immutability). A sanitizable signature scheme \textit{SSS} is immutable, iff for any efficient algorithm \(\mathcal{A}\) the probability that the experiment given in Fig. 9 returns 1 is negligible (as a function of \(\lambda\)) [7]. The basic idea is that an adversary generating the sanitizer key must be able to sanitize a block not designated to be sanitized. Note, the sanitizer key is created by the adversary. In other words, immutability is the unforgeability requirement for the sanitizer.

Definition 13 ((\textit{SSS}) Weak Immutability). A sanitizable signature scheme \textit{SSS} is weakly immutable, iff for any efficient algorithm \(\mathcal{A}\) the probability that the experiment given in Fig. 10 returns 1 is negligible (as a function of \(\lambda\)). The basic idea is that an adversary knowing the sanitizer key must be able to sanitize a block not designated to be sanitized. Note, the sanitizer key is not created by the sanitizer, only known.

Experiment WImmutability\textsuperscript{SSS}(\lambda)

\((pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \text{KeyGen}(1^\lambda)\)

\((pk_{\text{san}}, sk_{\text{san}}) \leftarrow \text{KeyGen}(1^\lambda)\)

\((m^*, \sigma^*, pk_{\text{san}}^*) \leftarrow \mathcal{A}^{\text{Sign}(\cdot, sk_{\text{sig}}, pk_{\text{san}}), \text{Proof}(sk_{\text{sig}}, \cdot, pk_{\text{san}}, \cdot)}(pk_{\text{sig}}, pk_{\text{san}}, sk_{\text{san}})\)

return 1, if:

\(\text{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk_{\text{san}}^*) = \text{true}\)

\(\exists i: \, m^*[j_i] \neq m_i[j_i]\) for some \(j_i \notin \text{ADM}_i\), or

\(pk_{\text{san}}^* \neq pk_{\text{san}}, i\)

shorter messages are padded with \(\bot\) or \(\text{ADM}_i \neq \text{ADM}^*\)

Fig. 10. Weak Immutability for \textit{SSS}

Interestingly, weak immutability is enough for our construction to be unforgeable, while for a \textit{SSS} used in the normal way, this definition is obviously not suitable at all.
2.3 Implications and Separations.

Let us formulate our first theorems:

**Theorem 1.** Every RSS which is strongly private, is also weakly private.

*Proof.* The game for strong privacy is less restrictive for the adversary than weak privacy. Hence, weak privacy is implied by strong privacy.

**Theorem 2.** There exists a RSS which is weakly private, but not strongly private.

*Proof.* See [18–20, 27] for examples. Additionally, we show that our scheme is not strongly private, i.e., only weakly, in App. A.

**Theorem 3.** Every SSS which is immutable is also weakly immutable.

*Proof.* The game for immutability is less restrictive for the adversary than weak immutability. Hence, immutability implies weak immutability.

**Theorem 4.** There exists a SSS which is private, but not strongly private.

*Proof.* We show this theorem by modifying an arbitrary existing strongly private SSS. Let $SSS = (\text{KGen}_{\text{sig}}, \text{KGen}_{\text{san}}, \text{Sign}, \text{Sanit}, \text{Verify}, \text{Proof}, \text{Judge})$ be an arbitrary private SSS. We alter the scheme as follows:

- $\text{KGen}_{\text{sig}}' := \text{KGen}_{\text{sig}}$, i.e., the key generation algorithm for the signer remains unchanged.
- $\text{KGen}_{\text{san}}' := \text{KGen}_{\text{san}}$, while an additional key pair for a IND-CCA2-secure encryption scheme [4] $\text{ENC}$ is generated.
- $\text{Sign}'$ is the same as $\text{Sign}$ with one exception; it appends the encryption $e$ of a digest of original message to the final signature, i.e., $\sigma' = (\sigma, e)$, where $e \leftarrow \text{ENC}(pk_{\text{san}}, H(m))$ and $H$ some standard cryptographic hash-function.
- $\text{Sanit}'$ is the same as $\text{Sanit}$ with one exception; it first removes the encrypted digest from the signature, while it appends it to the resulting signature.
- $\text{Verify}'$ is the same as $\text{Verify}$ with one exception; it removes the encrypted digest from the signature before verifying.
- $\text{Proof}'$ and $\text{Judge}'$ work essentially the same as their original counterparts, while cutting of $e$ from the signature before proceeding.

Clearly, a sanitizer, holding the corresponding secret key for $\text{ENC}$, can distinguish between messages generated by the signer and the sanitizer with overwhelming probability using the information contained in the signature $\sigma$. Without $sk_{\text{san}}$, this information remains hidden due to the IND-CCA2 encryption.
2.4 Definition of a Secure RSS and a Secure SSS.

We want to explicitly emphasize that accountability, as defined for SSSs in [7], has not been defined for RSSs yet, as Redact is a public algorithm. Hence, no secret sanitizer key(s) are required to allow any modifications. To circumvent this inconsistency, we utilize a standard SSS and let the signer generate the sanitizer key $sk_{san}$, attaching it to the signature, i.e., $\sigma' = (\sigma, sk_{san})$. If any alteration without $sk_{san}$ would be possible, the underlying SSS is obviously forgeable. As we have defined that this is a non-secure SSS, we omit this case. By doing so, the secret $sk_{san}$ becomes public knowledge and can be used by every party. This makes redacting a public operation. Hence, the secret key $sk_{san}$ for the sanitization becomes known to every party, including the signer to remain in the model defined for RSSs. This is the reason for our modifications of the existing security notions. We require these, on first sight very unnatural, restrictions to stay consistent with the standard model of SSSs as formalized in [7]. Moreover, the signer is generally not considered an adversarial entity in RSSs [33]. If other notions or adversary models are used, the results may obviously differ. In Sect. 4, we show that any SSS which only achieves standard privacy is not enough to construct a weakly private RSS and additional impossibility results. We show, without giving formal definitions, how one can derive an accountable RSS with explicitly denoted sanitizers in Sect. 5. How to formalize accountability for RSSs is an open question and is not answered within this paper.

As we aim to transform a SSS into a RSS, which by definition allows public redactions, we stick with the second approach, i.e., we require that $KGen_{san}$ is called by the signer and the resulting $sk_{san}$ is distributed by being publicly reconstructable from the signature. Since we have also dropped the requirement of an SSS to be accountable, we require weak blockwise non-interactive public accountability only for detecting (admissible) changes of blocks, twisting up the meaning its name implies.\(^1\) Vice versa, i.e., for the proof that no unforgeable RSS can be transformed into an unforgeable SSS, we neither require any additional security definitions nor any multi-sanitizer environments as introduced in [12]. In particular, we only need the unforgeability requirements of RSSs. Obviously, unforgeability is the most basic requirement, essential for every meaningful cryptographic construction.

We conclude this section with two final definitions:

**Definition 14 (Secure SSS).** We call a SSS secure, if, and only if, it is strongly private, weakly immutable, unforgeable and weakly blockwise non-interactive public accountable.

**Definition 15 (Secure RSS).** We call a RSS secure, if, and only if, it is weakly private and unforgeable.

\(^1\) To be more precise: [10] uses this feature to derive if the message has been sanitized: if this is the case, the sanitizer must be responsible and is therefore accountable
3 Generic Transformation of an SSS into an RSS

This section presents the generic transform. In particular, we give a generic algorithm to transform any unforgeable, strongly private, and weakly blockwise non-interactive publicly accountable SSS into an unforgeable and weakly private RSS.

Outline. The basic idea of our transform is that every party, including the signer, is allowed to alter all given blocks. The verification procedure accepts sanitized blocks, if, and only if, the altered blocks are ⊥. ⊥ is then be treated as a redacted block. Hence, redaction is altering a given block to a special symbol. As we have to defined that a SSS only allows strings $m[i] \in \{0, 1\}^*$, we need to define $\square := \emptyset$ and

$$m[i] \rightarrow \begin{cases} 0 & \text{if } m[i] = \emptyset \\ m[i] + 1 & \text{else} \end{cases}$$

to codify the additional symbol ⊥, where ⊥ expresses the empty string. Hence, we remain in the model defined. Moreover, this is where weak blockwise non-public interactive public accountability comes in: the changes to each block need to be detectable to allow a meaningful result, as a SSS allows arbitrary alterations. As ⊥ is still visible, the resulting scheme can only be weakly private, as statements about the original message $m$ can be made, contradicting our definition of strong privacy. Moreover, as a RSS allows that every party can redact blocks, we require that the sanitizing key $sk_{san}$ is known to every party, including the signer. Therefore, we need a strongly private SSS to achieve our definition of weak privacy for our RSS, as we prove in Sect. 4.

The Transform. In this section, we give a generic algorithm to actually perform the transform.

Construction 1 Let $SSS := (KGen_{sig}, KGen_{san}, \text{Sign}, \text{Sanit}, \text{Verify}, \text{Proof}, \text{Judge}, \text{Detect})$ be a secure SSS. Let the message space contain no □ symbol. Define the redactable signature scheme $RSS := (\text{KeyGen}, \text{Sign}, \text{Verify}, \text{Redact})$ as follows:

Key Generation: Algorithm $\text{KeyGen}$ generates on input of the security parameter $\lambda$, a key pair $(pk_{sig}, sk_{sig}) \leftarrow SSS.KGen_{sig}(1^\lambda)$ of the SSS, and also a sanitizer key pair $(pk_{san}, sk_{san}) \leftarrow SSS.KGen_{san}(1^\lambda)$. It returns $(sk, pk) = (sk_{sig}, (sk_{san}, pk_{san}, pk_{sig}))$

Signing: Algorithm $RSS.\text{Sign}$ on input $m \in \{0, 1\}^*$, sk, pk, sets ADM = $(1, \ldots, \ell)$ and computes $\sigma_s \leftarrow SSS.\text{Sign}(1^\lambda, m, sk_{sig}, pk_{san}, \text{ADM})$. It outputs: $(m, \sigma)$, where $\sigma = (sk_{san}, \sigma_s)$

Redacting: Algorithm $RSS.\text{Redact}$ on input message $m$, modification instructions MOD, a signature $\sigma = (sk_{san}, \sigma_s)$, keys $pk_{sig}$ and $pk_{san}$ first checks if
σ is a valid signature for m under the given public keys using RSS.Verify. If not, it stops outputting ⊥. Afterwards, it sets MOD' = {(i, □) | i ∈ MOD}. In particular, it generates a modification description for the SSS which sets block with index i ∈ MOD to □. Finally, it computes \( (m', \sigma'_s) \leftarrow SSS.Sanit(1^\lambda, m, MOD', \sigma_s, pk_{sig}, sk_{san}) \) and outputs \( (m', \sigma') \), where \( \sigma' = (sk_{san}, \sigma'_s) \).

Verification: Algorithm RSS.Verify on input a message \( m \in \{0, 1\}^* \), a signature \( \sigma = (sk_{san}, \sigma_s) \) and public keys \( pk_{sig}, pk_{san} \) first checks that ADM = (1, \ldots, ℓ) and that \( \sigma_s \) is a valid signature for \( m \) under the given public keys using SSS.Verify. If not, it returns false. Afterwards, for each i for which SSS.Detect(1^\lambda, m, \sigma_s, pk_{sig}, pk_{san}, i) returns San, it checks that \( m[i] = □ \). If not, it returns false. Else, it returns true. Note, one may also check if the given sanitizer key \( sk_{san} \) is correct.

Obviously, the secret sanitizer \( sk_{san} \) must be known to alter a block. That is the reason why it must be part of the signature to allow public redactions, as required by RSSs. Every redaction becomes visible as a message block containing the special symbol □.

**Theorem 5 (Our Construction is Secure).** If the utilized SSS is weakly blockwise non-interactive publicly accountable, weakly immutable and strongly private, the resulting RSS is weakly private and unforgeable, i.e., secure.

**Proof.** Th. 5 is proven in App. A.

**Theorem 6 (Our Construction is not Strongly Private).** Our construction is only weakly private, but not strongly private.

**Proof.** Due to Th. 5, we already know that our scheme is weakly private. Hence, it remains to show that it is not strongly private. As a redaction leaves a visible special symbol, i.e., □, an adversary can win the strong privacy game in the following way: Generate two messages \( m_0, m_1 \), where \( m_1 = (m_0, 1) \). Hence, \( ℓ_0 < ℓ_1 \), while \( m_0 \) is a prefix of \( m_1 \). Afterwards, it requests that \( m_1[i] = □ \) is redacted, i.e., MOD1 = (ℓ1) and MOD0 = (\). Hence, if the oracle chooses \( b = 0 \), it will output \( m_2 = m_0 \) and for \( b = 1 \), \( m_2 = (m_1, □) \). Obviously, the adversary can always win the game, as \( (m_1, □) \neq m_0 \).

Note, in the strong privacy game, ⊥ is not considered part of the message m. Hence, the scheme cannot be strongly private.

As RSSs allow every block to be removed, we require that ADM = (1, \ldots, ℓ). This rules out cases where a signer prohibits alterations of blocks, i.e., in our case, what we require for redaction. We show in Sect. 5 how this constraint can be transformed into the useful notion of consecutive disclosure control. In particular, there exists security models for RSSs, where prohibiting redactions is allowed and claimed to be useful, e.g., in [28, 37].
4 Minimum Requirements for a SSS to be Transformed

In this section, we show that standard private SSSs are not enough to build weakly private RSSs. Moreover, we prove that weak blockwise non-interactive public accountability is required to build an unforgeable RSS. To formally express this intuitive goal, we need the next theorems:

**Theorem 7 (Any non strongly private SSS, results in a non-weakly private RSS).** If the transformed SSS is only private, but not strongly private, the resulting RSS is not weakly private.

*Proof.* Let $A$ be an adversary winning the strong privacy game as defined in Fig. 5. We can then construct an adversary $B$, which wins the weak privacy game as defined in Fig. 6, using $A$ as a black-box in the following way:

1. The challenger generates $sk_{sig}, pk_{sig}, sk_{san}, pk_{san}$, and passes all but $sk_{sig}$ to $B$
2. $B$ passes all received keys to $A$. Note, $sk_{san}$ is required to for public redactions
3. $B$ simulates the signing oracle using the oracle provided by the challenger
4. Eventually, $A$ returns its guess $b^*$
5. $B$ outputs $b^*$ as its own guess

Following the definitions, the success probability of $A$ is non-negligible, the success probability of $B$ is non-negligible. In particular, the success probability of $B$ equals the one of $A$. This proves the theorem.

**Theorem 8 (No Transform can Result in a Strongly Private RSS).** There exists no algorithm which transforms a weakly immutable SSS into a strongly private RSS.

*Proof.* Once again, every meaningful SSS must be immutable, which implies weak immutability due to Th. 3. Hence, we do not make any statements about schemes not weakly immutable. We show that any transform $T$ achieving this property uses a $SSS'$ which is not weakly immutable. Also, our definition of a SSS requires that $ADM$ is always recoverable. Let $RSS'$ denote the resulting RSS. We can then derive an algorithm which uses $RSS'$ to break the weak immutability requirement of the underlying SSS in the following way:

1. The challenger generates the two key pairs of the SSS. It passes all keys but $sk_{sig}$ to $A$
2. $A$ transforms the $SSS$ into $RSS'$ given the transform $T$
3. $A$ calls the oracle $SSS.Sign$ with a message $m = (1, 2)$ and simulates $RSS'.Sign$
4. $A$ calls $RSS'.Redact$ with $MOD = (1)$
5. If the resulting signature $\sigma$ does not verify, abort

6. Output the resulting signature $\sigma_{SSS}$ of the underlying $SSS$

As $\ell_m \neq \ell \mod (m)$, $(\mod (m), \sigma_{SSS})$ breaks the weak immutability requirement of the $SSS$, as ADM is altered, contradicting our definition of a secure $SSS$. Moreover, as hiding redacted parts of a message is essential for strong privacy, no algorithm exists, which transforms a weakly immutable $SSS$ into a strongly private $RSS$, as ADM needs to be correctly recoverable. This proves the theorem. Note, we can give a concrete counterexample as we only use required behavior.

**Theorem 9 (Weak Blockwise Non-Interactive Accountability is Required for any Transform $T$).** For any transformation algorithm $T$, the utilized $SSS$ must be weakly blockwise non-interactive publicly accountable to result in an unforgeable $RSS$.

**Proof.** Let $RSS'$ be the resulting $RSS$. Perform the following steps to show that the resulting $RSS'$ is forgeable. In particular, let $A$ winning the weak blockwise non-interactive accountability game, which is used by $B$ to break the unforgeability of the resulting $RSS$.  

1. The challenger generates the two key pairs of the $SSS$. It passes all keys but $sk_{sig}$ to $B$  
2. $B$ forwards all received keys to $A$  
3. Any calls to the signing oracle by $A$ are answered genuinely by $B$ using its own signing oracle  
4. Eventually, $A$ returns a tuple $(m, \sigma_{SSS})$ to $B$  
5. If the resulting signature does not verify or does not win the weak blockwise non-interactive accountability game, $A$ and therefore also $B$ abort  
6. $A$ transforms the $SSS$ into $RSS'$ given the transform $T$  
7. If the resulting signature does not verify, $B$ aborts  
8. $B$ outputs the resulting $(m', \sigma_{RSS'})$ of the resulting $RSS'$

Following our definition in Fig. 3, $(m', \sigma_{RSS'})$ breaks the unforgeability requirement of the $RSS$, as there exists a block which has not been signed by the signer. Moreover, the success probabilities are equal.

**Theorem 10 (No Unforgeable RSS can be Transformed into a SSS).** There exists no transform $T$, which converts an unforgeable $RSS$ into a $SSS$.

**Proof.** Let $SSS'$ be the resulting $SSS$. Now perform the following steps to extract a valid forgery of the underlying $RSS$: 

1. The challenger generates a key pair for a $RSS$. It passes $pk$ to $A$.  

2. $A$ transforms $RSS$ into $SSS'$ given the transform $T$

3. $A$ calls the oracle $RSS$.Sign with a message $m = (1, 2)$ and simulates $SSS'$.Sign with $ADM = (1)$

4. $A$ calls $SSS'$.Sanit with $MOD = (1, a)$

5. If the resulting signature does not verify, abort

6. Output the resulting signature $\sigma_{RSS}$ of the underlying $RSS$

As $(a, 2) \notin \text{span}_w(m)$, $((a, 2), \sigma_{RSS})$ is a valid forgery of the underlying $RSS$. Note, this concrete counterexample is possible, as only required behavior is used.

5 Extensions

This section introduces additional modifications to our transform, which results in new properties not considered yet in previous works.

5.1 Consecutive Disclosure Control

We require $ADM = (1, \ldots, \ell)$, i.e. all message blocks are admissible to change. However, as already pointed out by Miyazaki et al. and Samelin et al., prohibiting consecutive redactions is a very useful feature [28, 29, 37]. In their case, the signer or an intermediate recipient is able to prohibit consecutive redaction. With our method, we achieve something different but related: we can prohibit that any consecutive party is able to prohibit sanitization. In particular, if $ADM \neq (1, \ldots, \ell)$, i.e., $\overline{ADM} = (1, \ldots, \ell) \setminus ADM$, a consecutive redaction is limited to blocks which are part of $ADM$. Obviously, this feature relies on the weak immutability property of the underlying $SSS$. Moreover, an intermediate recipient is able to remove the secret key $sk_{san}$ from the signature to prohibit any further redactions. To disallow this possibility, the sanitizing key must be signed and not just appended to the signature. If a different sanitizing key for each block is used, this allows a blockwise consecutive redaction control. We leave it as open work to formally define these properties.

5.2 Restricting to Sanitizers and Accountability

All $RSS$s allow everyone to redact blocks. To limit redaction to explicitly denoted sanitizers, the signature $\sigma$ is extended to hold an additional signature $\sigma_2$. Let $\sigma_2 \leftarrow \text{SIGN}(sk, CH(pk_{CH}, m))$, where $CH$ is a chameleon hash [23]. The values required to calculate $CH$ need to be delivered with $\sigma$. Hence, only sanitizers who possess the secret key $sk_{CH}$ for $CH$ can sanitize the message $m$ without invalidating the signature. This can be enriched further to achieve sanitizer and
signer accountability [7]: CH could be replaced with a tag-based chameleon-hash CH TAG, i.e., the construction of Brzuska et al. [7]. This idea has already been proposed by Samelin et al. [36, 37], but can also be applied for our scheme. However, a formal definition is still missing and thus as open work.

6 Conclusion

This paper presents how a SSS can be transformed into a RSS, if the corresponding security models are slightly adjusted. We gave a generic transform and proved the resulting RSS to be weakly private and unforgeable. Hence, all existing transforms are not suitable, as their security model is not strong enough to give sufficient privacy guarantees. We introduced the minimal set of security properties for an SSS that are required to yield a secure RSS. These strong notions have not been considered in previous work. Moreover, we give a rigorously argument that no RSS can be transformed into an unforgeable SSS. This implies, that SSSs and RSSs are completely different concepts.

It remains an open question, how to formally define accountability for RSSs and how RSSs and SSSs can be combined to yield a more flexible, yet fully private, sanitizable and redactable signature scheme. It also remains open, if unlinkable SSSs result in unlinkable RSSs.

References

A Proofs

Proof. This will prove Th. 5. We have to show that the resulting $RSS$ is unforgeable and weakly private. We prove each property on its own.
I) **Unforgeability.** Let $A$ be an algorithm breaking the unforgeability of the resulting $RSS$. We can then construct an algorithm $B$ which breaks the weak blockwise non-interactive publicly accountability of the utilized $SSS$. To do so, $B$ simulates $A$’s environment in the following way:

1. $B$ receives the following keys: $pk_{san}$, $sk_{san}$, $pk_{sig}$ and forwards them to $A$
2. For every query to the signing oracle, $B$ forwards the query to its own signing oracle and therefore is able to perfectly simulate the signing oracle for $A$
3. Eventually, $A$ outputs a tuple $(m^*, \sigma^*)$
4. If $(m^*, \sigma^*)$ does not verify or is trivial, abort

$B$ outputs $(m^*, \sigma^*)$ as its own forgery. Following the definition of unforgeability, $m$ cannot be derived from any queried message to the signature oracle, with the notable exception of $m[i] = \bot$ for any index $i$. Hence, there must exist at least one block $m[i] \neq \bot$, which has not been signed by the signer. Following our verification algorithm, the accepting verification requires that $\text{Sig} = \text{Detect}(1^\lambda, m^*, \sigma^*, pk_{sig}, pk_{san}, i)$. Hence, $(m^*, \sigma^*)$ breaks the weak blockwise non-interactive publicly accountability by outputting $(m^*, \sigma^*)$.

II) **Weak Privacy.** To show that our scheme is weakly private, we only need to show that an adversary $A$ can derive information about the prior content of a contained block $m[i]$, as $\bot$ is considered part of the resulting message $m'$ and all other modifications result in a forgeable $RSS$. Let $A$ winning the weak privacy game. We can then construct an adversary $B$ which breaks the strong privacy game in the following way:

1. $B$ receives the following keys: $pk_{san}$, $sk_{san}$, $pk_{sig}$ and forwards them to $A$
2. For every query to the signing oracle, $B$ forwards the query to its own signing oracle and therefore is able to perfectly simulate the signing oracle for $A$
3. $B$ also forwards any queries to its own $LoRSanit$ oracle. It passes the answers to $A$
4. Eventually, $A$ outputs its guess $b^*$

$B$ outputs $b^*$ as its own guess. As defined, the oracle requires that $\text{MOD}_1(m_1) = \text{MOD}(m_2)$, including any redacted blocks, i.e., $\bot$. In particular, the messages are the same. Hence, the success probability of $B$ is the same as $A$’s. This proves the theorem.