On the Relation Between Redactable and Sanitizable Signature Schemes

Hermann de Meer[1,3], Henrich C. Pöhls[2,3⋆], Joachim Posegga[2,3], Kai Samelin[4⋆⋆]

[1] Chair of Computer Networks and Computer Communication
[2] Chair of IT-Security
[3] Institute of IT-Security and Security Law (ISL), University of Passau, Germany
[4] Engineering Cryptographic Protocols Group & CASED, TU Darmstadt, Germany
demeer@uni-passau.de, {hp,jp}@sec.uni-passau.de, kai.samelin@ec-spride.de

Abstract. Malleable signature schemes ($\mathcal{MSS}$) enable a third party to alter signed data in a controlled way, maintaining a valid signature after an authorized change. Most well studied cryptographic constructions are (1) redactable signatures ($\mathcal{RSS}$), and (2) sanitizable signatures ($\mathcal{SSS}$). $\mathcal{RSS}$s allow the removal of blocks from a signed document, while $\mathcal{SSS}$s allow changing blocks to arbitrary strings. We rigorously prove that $\mathcal{RSS}$s are less expressive than $\mathcal{SSS}$s: no unforgeable $\mathcal{RSS}$ can be transformed into an $\mathcal{SSS}$. For the opposite direction we give a black-box transformation of a single $\mathcal{SSS}$, with tightened security, into an $\mathcal{RSS}$.

1 Introduction

Digital signatures are *the* IT-Security mechanism applied to detect integrity violations, as they become invalid on any change to the signed data. However, this also prohibits third parties from changing signed data in an allowed and controlled way. Applications where such an alteration is crucial, include secure routing [2] or "blank signatures" [23]. An additional prevailing reason to allow for subsequent changing or removing parts is the anonymization of personally identifiable information (PII), e.g., in medical data [28, 39]. Apart from the important privacy guarantee for the original data, it is often of paramount importance that the action of modification requires no additional interaction with the original signer. Hence, they are applicable in a wide area, e.g., in the Internet-of-Things (IoT) or for cloud computing [30]. For example, consider the IoT: communication

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

⋆⋆ Work partly carried out while working at the University of Passau, supported by "Regionale Wettbewerbsfähigkeit und Beschäftigung", Bayern, 2007-2013 (EFRE) as part of the SECBIT project (www.secbit.de) and the European Community's Seventh Framework Programme through the EINS Network of Excellence under grant agreement n° 288021. This work was also partly supported by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE and by the Hessian LOEWE excellence initiative within CASED.

with the originating sensor for re-signing would dramatically increase communication costs. Also in the case of smart metering privacy [16], the originating Smart meter must not know in what way the data it signed is later modified to preserve the user's privacy.

One cryptographically suitable approach to solve the above described "digital document sanitization problem" [34] are malleable signature schemes. Malleable signature schemes authorize certain changes to signed data such that the resulting changed message's authenticity is still verifiable, i.e., it remains verifiable that either none or only authorized changes have been applied to the signed message. This authorized change can come in several facets: Let $m = (m[1], \ldots, m[\ell])$, where $\ell \in \mathbb{N}$ and $m[i] \in \{0,1\}^*$, be a string m split up into ℓ parts we refer to as *blocks*. First, redactable signature schemes ($\mathcal{RSS}$) allow *anyone* to remove blocks $m[i]$ from m, without invalidating the signature. In particular, a redaction of the block $m[i]$, $0 < i \leq \ell$ leaves a blinded message m' without $m[i]$, i.e., $m' = (\ldots, m[i-1], \square, m[i+1], \ldots)$. A *visible* $\square$ has a major impact on the $\mathcal{RSS}$'s privacy guarantees. For $\mathcal{RSS}$s, it is required that *anyone* can derive a signature σ' which verifies for m'. Second, sanitizable signature schemes ($\mathcal{SSS}$), allow a sanitizer to change the *admissible* blocks, which are predefined by the signer, into arbitrary strings $m[i]' \in \{0,1\}^*$. Hence, the sanitizer can generate a verifiable message-signature pair (m', σ'). Contrary to $\mathcal{RSS}$s, in $\mathcal{SSS}$s the sanitizer holds its own secret key. Obviously, it must be verifiable that all changes were endorsed by the signer in both concepts.

Motivation. When more and more data containers are digitally signed as a countermeasure against attacks on their integrity and authenticity, it becomes increasingly important to be able to remove contained sensitive data with little impact on the integrity of the remaining data. In other words, signing data becomes the standard solution to allow integrity checks, e.g., for data stored on third-party storage servers in the cloud [30]. Current provably secure (unforgeable and private) solutions mostly focus on one out of two specific types of malleable signature schemes, i.e., either on $\mathcal{RSS}$s *or* $\mathcal{SSS}$s, often stating the other as related work, or even the same. At first sight, both approaches aim for the same goal, i.e., changing signed data. In detail, $\mathcal{SSS}$s only allow for *alterations* of blocks, while $\mathcal{RSS}$s only allow *removal of complete* blocks. Moreover, $\mathcal{SSS}$s require an additional key pair, while $\mathcal{RSS}$s have public redactions. The questions we answer in this paper is: What is the exact relation between both types of malleable signature?

Findings. We prove that an $\mathcal{RSS}$ is not *trivially* a "special case" of $\mathcal{SSS}$ [8, 41, 42]. But first things first: Obviously an unforgeable $\mathcal{SSS}$ can emulate a standard signature by disallowing any modifications by any sanitizer [42]. Second, we note that a $\mathcal{RSS}$ for a message of n blocks, fulfilling privacy [7], can trivially be constructed by deploying $\mathcal{O}(n^2)$ standard digital signatures [7, 37]. Hence, $\mathcal{O}(n^2)$ $\mathcal{SSS}$s are sufficient to construct one $\mathcal{RSS}$. Thus, from a theoretical point of view, $\mathcal{SSS}$s directly imply the existence of $\mathcal{RSS}$s. However, from a practical point of view, such constructions are rather inefficient. Especially as $\mathcal{RSS}$ can be constructed in $\mathcal{O}(n)$ for computation and storage [37]. We provide the security definitions required to transform only a *single* invocation of an $\mathcal{SSS}$ into an $\mathcal{RSS}$. We prove that the existing security models are not sufficient to achieve such a transformation. In particular, the resulting $\mathcal{RSS}$s cannot fulfill the state-of-the-art privacy definitions, as introduced in [7]. Note, for all transforms we treat $\mathcal{SSS}$ and $\mathcal{RSS}$ as *black-boxes* and ignore the constructions' details.

Contribution. This paper rigorously shows that $\mathcal{RSS}$s do not imply $\mathcal{SSS}$s: no unforgeable $\mathcal{RSS}$ can be transformed into a secure $\mathcal{SSS}$. While the converse is true in general, we give a more detailed separation: one *cannot* construct a *fully secure* $\mathcal{RSS}$ from a single $\mathcal{SSS}$ invocation, if one treats the used $\mathcal{SSS}$ as a black-box. In detail, weakening the privacy definition of $\mathcal{RSS}$s, while strengthening the security definitions of $\mathcal{SSS}$s, a single invocation of such a strong $\mathcal{SSS}$ can emulate a weaker $\mathcal{RSS}$. This paper provides an algorithm for a general transform: Any $\mathcal{SSS}$ that is (1) strongly private, (2) weakly immutable and (3) weakly blockwise non-interactive publicly accountable can be transformed into a weakly private $\mathcal{RSS}$. We give formal definitions of all the security properties in Sect. 2. Interestingly enough, it turns out that our definition of weak privacy is fulfilled by many existing $\mathcal{RSS}$s, which are considered not private following the model by *Brzuska* et al. [7].

Related Work. $\mathcal{SSS}$s have been introduced by *Ateniese* et al. [2] at ESORICS '05. *Brzuska* et al. formalized the most essential security properties [8]. These have later been extended for the properties of unlinkability [10, 12] and (block/groupwise) non-interactive public accountability [11, 18]. Moreover, several extensions and modifications like limiting-to-values [13, 27, 36], trapdoor $\mathcal{SSS}$s [15] and multi-sanitizer environments [14] have been considered.

$\mathcal{RSS}$s were introduced in 2002 by *Johnson* et al. in [26]. In the same year, *Steinfeld* and *Bull* introduced a similar concept as "Content Extraction Signatures" [40]. Since then, $\mathcal{RSS}$s have been subject to much research and got extended to tree-structured data [7, 28] and to arbitrary graphs [29]. *Samelin* et al. introduced the concept of redactable structure in [38]. The standard security properties of $\mathcal{RSS}$s have been formalized in [7, 17, 37]. *Ahn* et al. introduced the notion of context-hiding $\mathcal{RSS}$s [1]. Even stronger privacy notions have recently been introduced in [3, 4]. However, the scheme by *Ahn* et al. only achieves the less common notion of *selective* unforgeability [1]. Moreover, [1, 3, 4] are limited to quoting, i.e., redactions are only possible at the beginning, or end resp., of a list. There exists many additional work on $\mathcal{RSS}$s. We do note that most of the schemes are not fully private, e.g., [22, 24, 25, 31, 33]. Hence, a verifier can make statements about the original message m, which contradicts the intention of an $\mathcal{RSS}$ [7]. Most of these schemes achieve our notion of "weak privacy".

Combinations of both approaches appeared in [22, 24, 25]. However, their schemes do not preserve privacy [38]. While the work of *Yum* and *Joong* tries to combine the two properties in [42], the authors are not aware of any work considering relations between the notions.

Malleable signature schemes are usable in practice according to [35, 36].

We do note that there are also schemes aiming for calculating general functions on signed data, e.g., [5, 6, 19]. In this work, we focus on the relation between $\mathcal{SSS}$s and $\mathcal{RSS}$s.

2 Preliminaries and Security of $\mathcal{SSS}$ and $\mathcal{RSS}$

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

Sanitizable Signatures. The used notation is adapted from [8].

Definition 1 (Sanitizable Signature Scheme). *A $\mathcal{SSS}$ consists of at least seven efficient ($\mathcal{PPT}$) algorithms $\mathcal{SSS} := (\mathsf{KGen}_{sig}, \mathsf{KGen}_{san}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$:*

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 public key, using the security parameter λ:*
$(\mathrm{pk}_{sig}, \mathrm{sk}_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^\lambda), \quad (\mathrm{pk}_{san}, \mathrm{sk}_{san}) \leftarrow \mathsf{KGen}_{san}(1^\lambda)$

Signing. *The Sign algorithm takes $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$, the signer's secret key sk_{sig}, the sanitizer's public key pk_{san}, as well as a description $\textsc{adm}$ of the admissibly modifiable blocks, where $\textsc{adm}$ contains the number ℓ of blocks in m, as well the indices of the modifiable blocks. It outputs the message m and a signature σ (or $\perp$, indicating an error):*
$(m, \sigma) \leftarrow \mathsf{Sign}(1^\lambda, m, \mathrm{sk}_{sig}, \mathrm{pk}_{san}, \textsc{adm})$

Sanitizing. *Algorithm Sanit takes a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$, a signature σ, 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 $\textsc{mod}$, which contains pairs $(i, m[i]')$ for those blocks that shall be modified. Sanit calculates a new signature σ' for the modified message $m' \leftarrow \textsc{mod}(m)$. Then Sanit outputs m' and σ' (or $\perp$, indicating an error):*
$(m', \sigma') \leftarrow \mathsf{Sanit}(1^\lambda, m, \textsc{mod}, \sigma, \mathrm{pk}_{sig}, \mathrm{sk}_{san})$

Verification. *The Verify algorithm outputs a decision $d \in \{\mathtt{true}, \mathtt{false}\}$ verifying the validity of a signature σ for a message $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$ with respect to the public keys:*
$d \leftarrow \mathsf{Verify}(1^\lambda, m, \sigma, \mathrm{pk}_{sig}, \mathrm{pk}_{san})$

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 σ 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):*
$$\pi \leftarrow \mathsf{Proof}(1^\lambda, \mathrm{sk}_{sig}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \mathrm{pk}_{san})$$

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 σ, the public keys of the parties and a proof π. It outputs a decision $d \in \{\mathtt{Sig}, \mathtt{San}, \perp\}$ indicating whether the message-signature pair has been created by the signer or the sanitizer (or $\perp$, indicating an error):* $d \leftarrow \mathsf{Judge}(1^\lambda, m, \sigma, \mathrm{pk}_{sig}, \mathrm{pk}_{san}, \pi)$

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 [11]. The algorithm Detect is not part of the original $\mathcal{SSS}$ description by *Ateniese* et al., since it is not required for the purpose of a $\mathcal{SSS}$ [2, 8]. However, we require this algorithm later on to define (weak) blockwise non-interactive public accountability (See Def. 6).

Definition 2 ($\mathcal{SSS}$ Detect). *On input of the security parameter λ, a message-signature pair (m, σ), the corresponding public keys pk_{sig} and pk_{san}, and a block index $1 \le i \le \ell$, Detect outputs the accountable party (San or Sig) for block i (or $\perp$, indicating an error):*
$$d \leftarrow \mathsf{Detect}(1^\lambda, m, \sigma, \mathrm{pk}_{sig}, \mathrm{pk}_{san}, i), d \in \{\mathtt{San}, \mathtt{Sig}, \perp\}$$

We require the usual correctness properties to hold. In particular, all genuinely signed or sanitized messages are accepted, while every genuinely created proof π by the signer leads the judge to decide in favor of the signer. For a formal definition of correctness, refer to [8, 11]. It is also required by every $\mathcal{SSS}$ that ADM is always correctly recoverable from any valid message-signature pair (m, σ). This accounts for the work done in [21]. Jumping ahead, we want to emphasize that an $\mathcal{SSS}$ with weak non-interactive public accountability requires that Judge detects any *sanitization* on input of an empty proof $\pi = \perp$. Formal definitions of the security properties in a game-based manner follow.

Redactable Signatures. The following notation is derived from [38].

Definition 3 (Redactable Signature Schemes). *An $\mathcal{RSS}$ consists of four efficient algorithms $\mathcal{RSS} := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact})$:*

KeyGen. *The algorithm KeyGen outputs the public key* pk *and private key* sk *of the signer, where λ denotes the security parameter:*
$$(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$$

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

Verify. *The algorithm Verify outputs a decision $d \in \{\mathtt{true}, \mathtt{false}\}$, indicating the validity of the signature σ, w.r.t.* pk*, protecting $m = (m[1], \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$: $d \leftarrow \mathsf{Verify}(1^\lambda, \mathrm{pk}, m, \sigma)$*

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 σ and a*

```
**Experiment** $\mathsf{Unforgeability}_{\mathcal{RSS},\mathcal{A}}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda, sk, \cdot)}(pk)$
    let $i = 1, \ldots, q$ denote the queries to $\mathsf{Sign}$
  return 1, if
    $\mathsf{Verify}(1^\lambda, pk, m^*, \sigma^*) = 1$ and
    for all $i = 1, \ldots, q : m^* \notin \mathrm{span}_{\models}(m_i)$
```

Fig. 1. Unforgeability for $\mathcal{RSS}$

list of indices MOD *of blocks to be redacted. It returns a modified message* $m' \leftarrow \text{MOD}(m)$ *(or* $\perp$*, indicating an error):*

$$(m', \sigma') \leftarrow \mathsf{Redact}(1^\lambda, \mathrm{pk}, m, \sigma, \text{MOD})$$

We denote the transitive closure of m *as* $\mathit{span}_{\models}(m)$*. This set contains all messages derivable from* m *w.r.t.* Redact

As for $\mathcal{SSS}$s, the correctness properties for $\mathcal{RSS}$s are required to hold as well. Thus, every genuinely signed or redacted message must verify. Refer to [7] for a formal definition of correctness.

Security Models. This section contains the required security properties and models. They are derived from [8, 21, 38], but have been significantly altered. The requirement that ADM is always correctly reconstructible is captured within the unforgeability and immutability definitions. Note, following [8, 11, 12], an $\mathcal{SSS}$ must at least be unforgeable, immutable, accountable and private to be meaningful. Hence, we assume that all used $\mathcal{SSS}$s fulfill these four fundamental security requirements; if these requirements are not met, the construction is not considered an $\mathcal{SSS}$ and the results of this paper are not directly applicable. On the other hand, an $\mathcal{RSS}$ must be unforgeable and (weakly) private to be meaningful [7].

Unforgeability. No one should be able to compute a valid signature on a message not previously issued without having access to any private keys [7]. This is analogous to the unforgeability requirement for standard signature schemes [20], except that it excludes valid redactions from the set of forgeries for $\mathcal{RSS}$s, while for $\mathcal{SSS}$s *no* alterations are allowed.

Definition 4 ($\mathcal{RSS}$ Unforgeability). *We say that an* $\mathcal{RSS}$ *is unforgeable, if for any efficient (PPT) adversary* $\mathcal{A}$ *the probability that the game depicted in Fig. 1 returns* 1*, is negligible (as a function of* λ*).*

Definition 5 ($\mathcal{SSS}$ Unforgeability). *We say an* $\mathcal{SSS}$ *is unforgeable, if for any efficient (PPT) adversary* $\mathcal{A}$ *the probability that the game depicted in Fig. 2 returns* 1*, is negligible (as a function of* λ*).*

```
Experiment $\mathsf{Unforgeability}_{\mathcal{SSS},\mathcal{A}}(\lambda)$
  $(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KGen}_{\mathrm{sig}}(1^\lambda)$
  $(pk_{\mathrm{san}}, sk_{\mathrm{san}}) \leftarrow \mathsf{KGen}_{\mathrm{san}}(1^\lambda)$
  $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda,\cdot,sk_{\mathrm{sig}},\cdots)\mathsf{Proof}(1^\lambda,\cdot,sk_{\mathrm{sig}},\cdots),\mathsf{Sanit}(1^\lambda,\cdots,sk_{\mathrm{san}})}(pk_{\mathrm{sig}}, pk_{\mathrm{san}})$
    let $(m_i, \mathrm{ADM}_i, pk_{\mathrm{san},i})$ and $\sigma_i$ for $i = 1, 2, \ldots q$
    denote the queries/answers to/by the oracle $\mathsf{Sign}$,
    let $(m_j, \mathrm{MOD}_j, \sigma_j, pk_{\mathrm{sig},j})$ and $(m'_j, \sigma'_j)$ for $j = q + 1, \ldots, r$
    denote the queries/answers to/by the oracle $\mathsf{Sanit}$.
  return 1, if
    $\mathsf{Verify}(1^\lambda, m^*, \sigma^*, pk_{\mathrm{sig}}, pk_{\mathrm{san}}) = \mathtt{true}$ and
    for all $q = 1, \ldots q : (pk_{\mathrm{san}}, m^*, \mathrm{ADM}^*) \neq (pk_{\mathrm{san},i}, m_i, \mathrm{ADM}_i)$ and
    for all $j = q + 1, \ldots, r : (pk_{\mathrm{sig}}, m^*, \mathrm{ADM}^*) \neq (pk_{\mathrm{sig},j}, m_i, \mathrm{ADM}_i)$
```

Fig. 2. Unforgeability for $\mathcal{SSS}$

```
Experiment $\mathsf{WBlockPubAcc}_{\mathcal{SSS},\mathcal{A}}(\lambda)$
  $(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KGen}_{\mathrm{sig}}(1^\lambda)$
  $(pk_{\mathrm{san}}, sk_{\mathrm{san}}) \leftarrow \mathsf{KGen}_{\mathrm{san}}(1^\lambda)$
  $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda,\cdot,sk_{\mathrm{sig}},pk_{\mathrm{san}},\cdot),\mathsf{Proof}(1^\lambda,\cdot,sk_{\mathrm{sig}},\cdots,pk_{\mathrm{san}})}(pk_{\mathrm{san}}, sk_{\mathrm{san}}, pk_{\mathrm{sig}})$
    let $(m_i, \mathrm{ADM}_i)$ and $(m_i, \sigma_i)$ for $i = 1, \ldots, k$ be queries/answers to/by $\mathsf{Sign}$
    return 1, if
      $\mathsf{Verify}(1^\lambda, m^*, \sigma^*, pk_{\mathrm{sig}}, pk_{\mathrm{san}}) = \mathtt{true}$ and
      $\exists q$, s.t. $\mathsf{Detect}(1^\lambda, m^*, \sigma^*, pk_{\mathrm{sig}}, pk_{\mathrm{san}}, q) = \mathtt{Sig}$ and
      for all $i = 1, \ldots, k : (m^*[q], \sigma^*) \neq (m_i[q], \sigma_i)$.
  return 0
```

Fig. 3. Weak Blockwise Non-Interactive Public Accountability for $\mathcal{SSS}$

Weak Blockwise Non-Interactive Public Accountability. The basic idea is that an adversary, i.e., the sanitizer, has to be able to make the Detect algorithm accuse the signer, if it did not sign the specific block. Moreover, in our definition, the signer is *not* considered adversarial, contrary to *Brzuska* et al. [11]. An example for a weakly blockwise non-interactive publicly accountable $\mathcal{SSS}$ is the scheme introduced by *Brzuska* et al. [11]. We explain the reasons for our adversary model after the introduction of all required security properties. Note, pk_{san} is fixed for the oracles. For $\mathcal{SSS}$s, we also have sanitization and proof oracles [8].

Definition 6 ($\mathcal{SSS}$ Weak Blockwise Non-Interactive Public Accountability). *A sanitizable signature scheme $\mathcal{SSS}$ is* weakly non-interactive publicly accountable*, if $\mathsf{Proof} = \perp$, 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 λ).*

Privacy. No one should be able to gain any knowledge about sanitized parts without having access to them [8]. This is similar to the standard indistinguishability notion for encryption schemes. The basic idea is that the oracle either signs and sanitizes the first message (m_0) or the second (m_1), while the resulting message must be the same for each input. The adversary must not be able to decide which input message was used.

```
$\textbf{Experiment } \mathsf{Privacy}_{\mathcal{SSS},\mathcal{A}}(\lambda)$
  $(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
  $(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^\lambda)$
  $b \leftarrow \{0,1\}$
  $a \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda, sk_{\text{sig}},\cdots),\mathsf{Proof}(1^\lambda, sk_{\text{sig}},\cdots),\mathsf{LoRSanit}(\cdots,sk_{\text{sig}},sk_{\text{san}},b),\mathsf{Sanit}(1^\lambda,\cdots,sk_{\text{san}})}(pk_{\text{sig}}, pk_{\text{san}})$
    where oracle $\mathsf{LoRSanit}$ 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 $\perp$
    if $\text{MOD}_0 \not\subseteq \text{ADM} \vee \text{MOD}_1 \not\subseteq \text{ADM}$, return $\perp$
    let $(m, \sigma) \leftarrow \mathsf{Sign}(1^\lambda, m_b, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})$
    return $(m', \sigma') \leftarrow \mathsf{Sanit}(1^\lambda, m, \text{MOD}_b, \sigma, pk_{\text{sig}}, sk_{\text{san}})$
  return 1, if $a = b$
```

Fig. 4. Standard Privacy for $\mathcal{SSS}$

Definition 7 ($\mathcal{SSS}$ Standard Privacy). *We say that an $\mathcal{SSS}$ is (standard) private, if for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 4 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ).*

The aforementioned privacy definition [8] only considers outsiders as adversarial. However, we require that even insiders, i.e., sanitizers, are not able to win the game. Note, the key sk_{san} is *not* generated by the adversary, only known to it. We explain the need for this alteration after the next definitions. For our definition of 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_{san} of the sanitizer. This notion extends the definition of standard privacy (Fig. 4) to also account for parties knowing the secret sanitizer key sk_{san}. In a sense, this definition captures some form of "forward-security". Examples for strongly private $\mathcal{SSS}$s are the schemes introduced by *Brzuska* et al. [9, 11, 12], as their schemes are perfectly private. As the adversary now knows sk_{san}, it can trivially simulate the sanitization oracle itself.

Definition 8 ($\mathcal{SSS}$ Strong Privacy). *We say that an $\mathcal{SSS}$ is private, if for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 5 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ).*

In a weakly private $\mathcal{RSS}$, a third party can derive which parts of a message have been redacted without gathering more information, as redacted blocks are replaced with $\square$, which is visible. The basic idea is that the oracle either signs and sanitizes the first message (m_0) or the second (m_1). 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 $\square$ is considered part of the message. For strong privacy, this constraint is not required. We want to emphasize, that *Lim* et al. define weak privacy in a different manner: they prohibit access to the signing oracle [31]. Our definition allows for such adaptive

```
Experiment $\mathsf{SPrivacy}_{\mathcal{SSS},\mathcal{A}}(\lambda)$
  $(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KGen}_{\mathrm{sig}}(1^\lambda)$
  $(pk_{\mathrm{san}}, sk_{\mathrm{san}}) \leftarrow \mathsf{KGen}_{\mathrm{san}}(1^\lambda)$
  $b \leftarrow \{0,1\}$
  $a \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda,\cdot,sk_{\mathrm{sig}},pk_{\mathrm{san}},\cdot),\mathsf{Proof}(1^\lambda,sk_{\mathrm{sig}},\cdots,pk_{\mathrm{san}}),\mathsf{LoRSanit}(\cdots,sk_{\mathrm{sig}},sk_{\mathrm{san}},b)}(pk_{\mathrm{sig}}, pk_{\mathrm{san}}, sk_{\mathrm{san}})$
    where oracle $\mathsf{LoRSanit}$ on input of:
    $m_0, \mathrm{MOD}_0, m_1, \mathrm{MOD}_1, \mathrm{ADM}$
    if $\mathrm{MOD}_0(m_0) \neq \mathrm{MOD}_1(m_1)$, return $\perp$
    if $\mathrm{MOD}_0 \not\subseteq \mathrm{ADM} \vee \mathrm{MOD}_1 \not\subseteq \mathrm{ADM}$, return $\perp$
    let $(m,\sigma) \leftarrow \mathsf{Sign}(1^\lambda, m_b, sk_{\mathrm{sig}}, pk_{\mathrm{san}}, \mathrm{ADM})$
    return $(m',\sigma') \leftarrow \mathsf{Sanit}(1^\lambda, m, \mathrm{MOD}_b, \sigma, pk_{\mathrm{sig}}, sk_{\mathrm{san}})$
  return 1, if $a = b$
```

Fig. 5. Strong Privacy for $\mathcal{SSS}$

```
Experiment $\mathsf{WPrivacy}_{\mathcal{RSS},\mathcal{A}}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $b \leftarrow \{0,1\}$
  $d \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda,sk,\cdot),\mathsf{LoRRedact}(\cdots,sk,b)}(pk)$
    where oracle $\mathsf{LoRRedact}$
    for input $m_0, m_1, \mathrm{MOD}_0, \mathrm{MOD}_1$:
    if $\mathrm{MOD}_0(m_0) \neq \mathrm{MOD}_1(m_1)$, return $\perp$
    Note: redacted blocks are denoted $\square$, which are considered part of $m$
    $(m,\sigma) \leftarrow \mathsf{Sign}(1^\lambda, sk, m_b)$
    return $(m',\sigma') \leftarrow \mathsf{Redact}(1^\lambda, pk, m, \sigma, \mathrm{MOD}_b)$.
  return 1, if $b = d$
```

Fig. 6. Weak Privacy for $\mathcal{RSS}$

queries. Summarized, weak privacy only makes statements about blocks, not the complete message. See [28] for possible attacks. Weakly private schemes, following our definition, are, e.g., [22, 28]. In their schemes, the adversary is able to pinpoint the indices of the redacted blocks, as $\square$ is visible.

Definition 9 ($\mathcal{RSS}$ Weak Privacy). *We say that an $\mathcal{RSS}$ is weakly private, if for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 6 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ).*

The next definition is similar to weak privacy. However, redacted parts are *not* considered part of the message.

Definition 10 ($\mathcal{RSS}$ Strong Privacy). *We say that an $\mathcal{RSS}$ is strongly private, if for any efficient (PPT) adversary $\mathcal{A}$ the probability that the game depicted in Fig. 7 returns 1, is negligibly close to $\frac{1}{2}$ (as a function of λ). This is the standard definition of privacy [7].*

Immutability. The idea behind immutability is that an adversary generating the sanitizer key must only be able to sanitize admissible blocks. Hence, immutability is the unforgeability requirement for the sanitizer.

```
**Experiment** $\mathsf{SPrivacy}_{\mathcal{RSS},\mathcal{A}}(\lambda)$
  $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $b \leftarrow \{0,1\}$
  $d \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda, sk, \cdot), \mathsf{LoRRedact}(\cdots, sk, b)}(pk)$
    where oracle $\mathsf{LoRRedact}$
    for input $m_0, m_1, \mathrm{MOD}_0, \mathrm{MOD}_1$:
    if $\mathrm{MOD}_0(m_0) \neq \mathrm{MOD}_1(m_1)$, return $\perp$
    Note: redacted blocks are *not* considered part of the message
    $(m, \sigma) \leftarrow \mathsf{Sign}(1^\lambda, sk, m_b)$
    return $(m', \sigma') \leftarrow \mathsf{Redact}(1^\lambda, pk, m, \sigma, \mathrm{MOD}_b)$.
  return 1, if $b = d$
```

Fig. 7. Strong Privacy for $\mathcal{RSS}$

```
**Experiment** $\mathsf{Immutability}_{\mathcal{SSS},\mathcal{A}}(\lambda)$
  $(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $(m^*, \sigma^*, pk^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda, \cdot, sk_{\mathrm{sig}}, \cdot, \cdot), \mathsf{Proof}(1^\lambda, sk_{\mathrm{sig}}, \cdots)}(pk_{\mathrm{sig}})$
  let $(m_i, \mathrm{ADM}_i, pk_{\mathrm{san},i})$ and $\sigma_i$ for $i = 1, \ldots, q$ be queries/answers to/by $\mathsf{Sign}$
  return 1, if:
    $\mathsf{Verify}(1^\lambda, m^*, \sigma^*, pk_{\mathrm{sig}}, pk^*) = \mathtt{true}$ and
    for all $i = 1, 2, \ldots, q : (pk^*, m^*[j_i], \mathrm{ADM}^*) \neq (pk_{\mathrm{san},i}, m_i[j_i], \mathrm{ADM}_i)$ and
    if $(m^*[j_i], \mathrm{ADM}_i, pk_{\mathrm{san},i}) \neq (m_i[j_i], \mathrm{ADM}_i, pk_{\mathrm{san},i})$, also $j_i \notin \mathrm{ADM}_i$
    where shorter messages are padded with $\perp$
```

Fig. 8. Immutability for $\mathcal{SSS}$

Definition 11 ($\mathcal{SSS}$ Immutability). *A sanitizable signature scheme $\mathcal{SSS}$ is immutable, if for any efficient algorithm $\mathcal{A}$ the probability that the experiment from Fig. 8 returns* 1 *is negligible (as a function of λ) [8].*

For weak immutability, an adversary knowing, but not generating, the sanitizer key must only be able to sanitize admissible blocks. Hence, once more, pk_{san} is fixed.

Definition 12 ($\mathcal{SSS}$ Weak Immutability). *A sanitizable signature scheme $\mathcal{SSS}$ is weakly immutable, if for any efficient algorithm $\mathcal{A}$ the probability that the experiment given in Fig. 9 returns* 1 *is negligible (as a function of λ).*

Interestingly, weak immutability is enough for our construction to be unforgeable, while for an $\mathcal{RSS}$ used in the normal way, this definition is obviously not suitable at all due to accountability reasons. We omit the security parameter λ for the rest of the paper to increase readability.

Implications and Separations. Let us formulate our first theorems:

Theorem 1. *There exists an $\mathcal{RSS}$ which is only weakly private.*

```
Experiment $\mathsf{WImmutability}_{\mathcal{SSS},\mathcal{A}}(\lambda)$
  $(pk_{\mathrm{sig}}, sk_{\mathrm{sig}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $(pk_{\mathrm{san}}, sk_{\mathrm{san}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(1^\lambda,\cdot,sk_{\mathrm{sig}},pk_{\mathrm{san}},\cdot),\mathsf{Proof}(1^\lambda,sk_{\mathrm{sig}},\cdots,pk_{\mathrm{san}},\cdot)}(pk_{\mathrm{sig}}, pk_{\mathrm{san}}, sk_{\mathrm{san}})$
  let $(m_i, \mathrm{ADM}_i)$ and $\sigma_i$ for $i = 1, 2, \ldots q$ be queries/answers to/by Sign
  return 1, if:
    $\mathsf{Verify}(1^\lambda, m^*, \sigma^*, pk_{\mathrm{sig}}, pk_{\mathrm{san}}) = \mathtt{true}$ and
    $\forall i, i = 1, 2, \ldots, q : (m^*[j_i], \mathrm{ADM}^*) \neq (m_i[j_i], \mathrm{ADM}_i)$ and
    if $(m^*[j_i], \mathrm{ADM}_i) \neq (m_i[j_i], \mathrm{ADM}_i)$, also $j_i \notin \mathrm{ADM}_i$
    where shorter messages are padded with $\perp$
```

Fig. 9. Weak Immutability for $\mathcal{SSS}$

Proof. See [22, 24, 25, 31] for examples.

Theorem 2. *Every $\mathcal{SSS}$ which is immutable, is also weakly immutable.*

Proof. Trivially implied: $\mathcal{A}$ generates the sanitizer key pair honestly.

Theorem 3. *There exists an $\mathcal{SSS}$ which is private, but not strongly private.*

Th. 3 is proven in App. B.

Definition of a Secure $\mathcal{RSS}$ and a Secure $\mathcal{SSS}$. We want to explicitly emphasize that accountability, as defined for $\mathcal{SSS}$s in [8], has not been defined for $\mathcal{RSS}$s yet, as Redact is a public algorithm. Hence, no secret sanitizer key(s) are required for redactions. To circumvent this inconsistency, we utilize a standard $\mathcal{SSS}$ and let the signer generate the sanitizer key sk_{san}, attaching it to the public key of the signer. This also explains why pk_{san} is fixed in our security model. If any alteration without sk_{san} is possible, the underlying $\mathcal{SSS}$ would obviously be forgeable. As we have defined that this is a non-secure $\mathcal{SSS}$, we omit this case. Hence, the secret sk_{san} becomes public knowledge and can be used by every party. This is the reason why the adversary only knows sk_{sig}, but cannot generate it. We require these, at first sight very unnatural, restrictions to stay consistent with the standard model of $\mathcal{SSS}$s as formalized in [8]. Moreover, the signer is generally *not* considered an adversarial entity in $\mathcal{RSS}$s [7]. If other notions or adversary models are used, the results may obviously differ. In App. A, we show that any $\mathcal{SSS}$ which only achieves standard privacy, is not enough to construct a weakly private $\mathcal{RSS}$ and additional impossibility results.

3 Generic Transformation

This section presents the generic transform. In particular, we provide a generic algorithm which transforms any weakly immutable, strongly private, and weakly blockwise non-interactive publicly accountable $\mathcal{SSS}$ into an unforgeable and weakly private $\mathcal{RSS}$.

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 the altered blocks are $\square$. $\square$ is treated as a redacted block. Hence, redaction is altering a given block to a special symbol. As we have defined that an $\mathcal{SSS}$ only allows for strings $m[i] \in \{0,1\}^*$, we need to define $\square := \emptyset$ and $m[i] \leftarrow 0$, if $m[i] = \emptyset$ and $m[i] \leftarrow m[i] + 1$ else to codify the additional symbol $\square$. Here, $\emptyset$ 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 for a meaningful result, as an $\mathcal{SSS}$ allows for arbitrary alterations. As $\square$ is still visible, the resulting scheme is only weakly private, as statements about m can be made. This contradicts our definition of strong privacy for $\mathcal{RSS}$s. Moreover, as an $\mathcal{RSS}$ allows every party to redact blocks, it is obvious that sk_{san} must be known to every party, including the signer. Therefore, we need a strongly private $\mathcal{SSS}$ to achieve our definition of weak privacy for the $\mathcal{RSS}$, as proven in App. A.

Construction 1 *Let* $\mathcal{SSS} := (\mathsf{KGen}_{sig}, \mathsf{KGen}_{san}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge}, \mathsf{Detect})$ *be a secure* $\mathcal{SSS}$*. Define* $\mathcal{RSS} := (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Redact})$ *as follows:*

Key Generation: Algorithm KeyGen *generates on input of the security parameter* λ*, a key pair* $(\mathrm{pk}_{sig}, \mathrm{sk}_{sig}) \leftarrow \mathcal{SSS}.\mathsf{KGen}_{sig}(1^\lambda)$ *of the* $\mathcal{SSS}$*, and also a sanitizer key pair* $(\mathrm{pk}_{san}, \mathrm{sk}_{san}) \leftarrow \mathcal{SSS}.\mathsf{KGen}_{san}(1^\lambda)$*. It returns* $(\mathrm{sk}, \mathrm{pk}) = (\mathrm{sk}_{sig}, (\mathrm{sk}_{san}, \mathrm{pk}_{san}, \mathrm{pk}_{sig}))$

Signing: Algorithm $\mathcal{RSS}.\mathsf{Sign}$ *on input* $m \in \{0,1\}^*, \mathrm{sk}, \mathrm{pk}$*, sets* $\text{ADM} = (1, \dots, \ell)$ *and computes* $\sigma \leftarrow \mathcal{SSS}.\mathsf{Sign}(1^\lambda, m, \mathrm{sk}_{sig}, \mathrm{pk}_{san}, \text{ADM})$*. It outputs:* (m, σ)

Redacting: Algorithm $\mathcal{RSS}.\mathsf{Redact}$ *on input message* m*, modification instructions* MOD*, a signature* σ*, keys* $\mathrm{pk} = (\mathrm{sk}_{san}, \mathrm{pk}_{san}, \mathrm{pk}_{sig})$*, first checks if* σ *is a valid signature for* m *under the given public keys using* $\mathcal{RSS}.\mathsf{Verify}$*. If not, it stops outputting* $\perp$*. Afterwards, it sets* $\text{MOD}' = \{(i, \square) \mid i \in \text{MOD}\}$*. In particular, it generates a modification description for the* $\mathcal{SSS}$ *which sets block with index* $i \in \text{MOD}$ *to* $\square$*. Finally, it outputs* $(m', \sigma') \leftarrow \mathcal{SSS}.\mathsf{Sanit}(1^\lambda, m, \text{MOD}', \sigma, \mathrm{pk}_{sig}, \mathrm{sk}_{san})$

Verification: Algorithm $\mathcal{RSS}.\mathsf{Verify}$ *on input a message* $m \in \{0,1\}^*$*, a signature* σ *and* pk *first checks that* $\text{ADM} = (1, \dots, \ell)$ *and that* σ *is a valid signature for* m *under the given public keys using* $\mathcal{SSS}.\mathsf{Verify}$*. If not, it returns* $\mathtt{false}$*. Afterwards, for each* i *for which* $\mathcal{SSS}.\mathsf{Detect}(1^\lambda, m, \sigma_s, \mathrm{pk}_{sig}, \mathrm{pk}_{san}, i)$ *returns* San*, it checks that* $m[i] = \square$*. If not, it returns* $\mathtt{false}$*. Else, it returns* $\mathtt{true}$*. One may also check, if* sk_{san} *is correct and that all* $m[i]$ *are sanitizable, if required.*

Theorem 4 (Our Construction is Secure). *If the utilized* $\mathcal{SSS}$ *is weakly blockwise non-interactive publicly accountable, weakly immutable and strongly private, the resulting* $\mathcal{RSS}$ *is weakly private, but not strongly, and unforgeable.*

Th. 4 is proven in App. B.

As $\mathcal{RSS}$s allow for removing every block, we require that $\textsc{adm} = (1, \ldots, \ell)$. This rules out cases where a signer prohibits alterations of blocks. This constraint can easily be transformed into the useful notion of consecutive disclosure control [32, 38].

4 Conclusion and Future Work

This paper presents a method to transform a single instantiation of an $\mathcal{SSS}$ into an $\mathcal{RSS}$. In detail, if we use one $\mathcal{SSS}$ instantiation, an emulation of an $\mathcal{RSS}$ can only be achieved, if the $\mathcal{SSS}$'s security is strengthened, raising it above the existing standard. The resulting emulated $\mathcal{RSS}$ offers only weaker privacy guarantees. Moreover, we have argued rigorously that the opposite implication is not possible. Thus, no $\mathcal{RSS}$ can be transformed into an unforgeable $\mathcal{SSS}$. Hence, $\mathcal{RSS}$s and $\mathcal{SSS}$s are indeed two different cryptographic building blocks, even if they achieve to define and delegate authorized modifications of signed messages. Currently, the number of $\mathcal{SSS}$s achieving the new security requirements needed to securely emulate an $\mathcal{RSS}$ is still low.

For the future, we suggest to focus on implementing and standardizing an $\mathcal{SSS}$ secure enough to emulate $\mathcal{RSS}$s, to have one universal building block. In the meantime we advice to use dedicated $\mathcal{RSS}$ algorithms if only redactions are needed and a $\mathcal{SSS}$ algortihm. Of course, you are advised to check current work to ensure the cryptographic strength of the constructions.

Cryptographically, remaining open questions are: how to formally define accountability for $\mathcal{RSS}$s, to identify if the interesting privacy properties of unlinkability for $\mathcal{SSS}$ [10, 12] will carry forward when transformed into an $\mathcal{RSS}$, and to further research how $\mathcal{RSS}$ and $\mathcal{SSS}$s can be combined.

References

1. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. Computing on authenticated data. Cryptology ePrint Archive, Report 2011/096, 2011. http://eprint.iacr.org.
2. G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS*, pages 159–177. Springer, 2005.
3. N. Attrapadung, B. Libert, and T. Peters. Computing on authenticated data: New privacy definitions and constructions. In *ASIACRYPT*, pages 367–385, 2012.
4. N. Attrapadung, B. Libert, and T. Peters. Efficient completely context-hiding quotable and linearly homomorphic signatures. In *PKC*, pages 386–404, 2013.
5. D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149–168, 2011.
6. D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *Public Key Cryptography*, pages 1–16, 2011.
7. C. Brzuska, H. Busch, O. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In *ACNS*, ACNS'10, pages 87–104. Springer, 2010.

8. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of Sanitizable Signatures Revisited. In *PKC 2009*, pages 317–336. Springer, 2009.
9. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Sanitizable signatures: How to partially delegate control for authenticated data. In *Proc. of BIOSIG*, volume 155 of *LNI*, pages 117–128. GI, 2009.
10. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of Sanitizable Signatures. In *PKC*, pages 444–461, 2010.
11. C. Brzuska, H. C. Pöhls, and K. Samelin. Non-Interactive Public Accountability for Sanitizable Signatures. In *EuroPKI*, volume 7868 of *LNCS*, pages 178–193. Springer, 2012.
12. C. Brzuska, H. C. Pöhls, and K. Samelin. Efficient and Perfectly Unlinkable Sanitizable Signatures without Group Signatures. In *EuroPKI 2013*, volume 8341 of *LNCS*, pages 12–30. Springer, 2014.
13. S. Canard and A. Jambert. On extended sanitizable signature schemes. In *CT-RSA*, pages 179–194, 2010.
14. S. Canard, A. Jambert, and R. Lescuyer. Sanitizable signatures with several signers and sanitizers. In *AFRICACRYPT*, pages 35–52, 2012.
15. S. Canard, F. Laguillaumie, and M. Milhau. Trapdoor sanitizable signatures and their application to content protection. In *ACNS*, pages 258–276, 2008.
16. A. Cavoukian, J. Polonetsky, and C. Wolf. Smartprivacy for the smart grid: embedding privacy into the design of electricity conservation. *Identity in the Information Society*, 3(2):275–294, 2010.
17. E.-C. Chang, C. L. Lim, and J. Xu. Short Redactable Signatures Using Random Trees. In *CT-RSA*, CT-RSA '09, pages 133–147, Berlin, Heidelberg, 2009. Springer.
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. D. M. Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *PKC*, pages 697–714, 2012.
20. S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, 17:281–308, 1988.
21. J. Gong, H. Qian, and Y. Zhou. Fully-secure and practical sanitizable signatures. In *InsCrypt*, volume 6584 of *LNCS*, pages 300–317. Springer, 2011.
22. 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.
23. C. Hanser and D. Slamanig. Blank digital signatures. In *AsiaCCS*, pages 95 – 106. ACM, 2013.
24. T. Izu, M. Izumi, N. Kunihiro, and K. Ohta. Yet another sanitizable and deletable signatures. In *AINA*, pages 574–579, 2011.
25. T. Izu, N. Kunihiro, K. Ohta, M. Sano, and M. Takenaka. Sanitizable and deletable signature. In *ISA*, volume 5379 of *LNCS*, pages 130–144. Springer, 2009.
26. R. Johnson, D. Molnar, D. Song, and D.Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262. Springer, Feb. 2002.
27. M. Klonowski and A. Lauks. Extended Sanitizable Signatures. In *ICISC*, pages 343–355, 2006.
28. A. Kundu and E. Bertino. Structural Signatures for Tree Data Structures. In *Proc. of PVLDB 2008*, New Zealand, 2008. ACM.
29. A. Kundu and E. Bertino. How to authenticate graphs without leaking. In *EDBT*, pages 609–620, 2010.

30. A. Kundu and E. Bertino. Privacy-preserving authentication of trees and graphs. *Intl. J. of Inf. Sec.*, pages 1–28, 2013.
31. S. Lim, E. Lee, and C.-M. Park. A short redactable signature scheme using pairing. *Sec. and Comm. Netw.*, 5(5):523–534, 2012.
32. K. Miyazaki, G. Hanaoka, and H. Imai. Digitally signed document sanitizing scheme based on bilinear maps. ASIACCS '06, pages 343–354, New York, NY, USA, 2006. ACM.
33. 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.
34. K. Miyazaki, S. Susaki, M. Iwamura, T. Matsumoto, R. Sasaki, and H. Yoshiura. Digital documents sanitizing problem. Technical report, IEICE, 2003.
35. H. C. Pöhls, S. Peters, K. Samelin, J. Posegga, and H. de Meer. Malleable signatures for resource constrained platforms. In *WISTP*, pages 18–33, 2013.
36. H. C. Pöhls, K. Samelin, and J. Posegga. Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In *ACNS*, volume 6715 of *LNCS*, pages 166–182. Springer, 2011.
37. K. Samelin, H. C. Pöhls, A. Bilzhause, J. Posegga, and H. de Meer. On Structural Signatures for Tree Data Structures. In *ACNS*, volume 7341 of *LNCS*, pages 171–187. Springer, 2012.
38. K. Samelin, H. C. Pöhls, A. Bilzhause, J. Posegga, and H. de Meer. Redactable signatures for independent removal of structure and content. In *ISPEC*, volume 7232 of *LNCS*, pages 17–33. Springer, 2012.
39. D. Slamanig and S. Rass. Generalizations and extensions of redactable signatures with applications to electronic healthcare. In *Communications and Multimedia Security*, pages 201–213, 2010.
40. R. Steinfeld and L. Bull. Content extraction signatures. In *ICISC*. Springer Berlin / Heidelberg, 2002.
41. T. H. Yuen, W. Susilo, J. K. Liu, and Y. Mu. Sanitizable signatures revisited. In *CANS*, pages 80–97, 2008.
42. D. H. Yum, J. W. Seo, and P. J. Lee. Trapdoor sanitizable signatures made easy. In *ACNS*, ACNS'10, pages 53–68, Berlin, Heidelberg, 2010. Springer.

A Requirements to Transform a $\mathcal{SSS}$ into a $\mathcal{RSS}$

In this section, we show that standard private $\mathcal{SSS}$s are not enough to build weakly private $\mathcal{RSS}$s. Moreover, we prove that weak blockwise non-interactive public accountability is required to build an unforgeable $\mathcal{RSS}$. To formally express this intuitive goals, we need Theorems 5 and 6.

Theorem 5 (Any non strongly private $\mathcal{SSS}$ results in a non-weakly private $\mathcal{RSS}$). *If the transformed $\mathcal{SSS}$ is not strongly private, the resulting $\mathcal{RSS}$ is not weakly private.*

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

1. $\mathcal{B}$ receives the following keys from the challenger: $pk_{\text{san}}, sk_{\text{san}}, pk_{\text{sig}}$ and forwards them to $\mathcal{A}$

2. $\mathcal{B}$ simulates the signing oracle using the oracle provided
3. Eventually, $\mathcal{A}$ returns its guess b^*
4. $\mathcal{B}$ outputs b^* as its own guess

Following the definitions, the success probability of $\mathcal{B}$ equals the one of $\mathcal{A}$. This proves the theorem.

Theorem 6 (No Transform can Result in a Strongly Private $\mathcal{RSS}$). *There exists no algorithm which transforms a secure $\mathcal{SSS}$ into a strongly private $\mathcal{RSS}$.*

Proof. Once again, every meaningful $\mathcal{SSS}$ must be immutable, which implies weak immutability due to Th. 2. Hence, we do not make any statements about schemes not weakly immutable. We show that any transform $\mathcal{T}$ achieving this property uses a $\mathcal{SSS}'$ which is not weakly immutable. Let $\mathcal{RSS}'$ denote the resulting $\mathcal{RSS}$. We can then derive an algorithm which uses $\mathcal{RSS}'$ to break the weak immutability requirement of the underlying $\mathcal{SSS}$ in the following way:

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

As $\ell_m \neq \ell\text{MOD}(m)$, $(\textsc{mod}(m), \sigma_{\mathcal{SSS}})$ breaks the weak immutability requirement of the $\mathcal{SSS}$. Moreover, as hiding redacted parts of a message is essential for strong privacy, no algorithm exists, which transforms a weakly immutable $\mathcal{SSS}$ into a strongly private $\mathcal{RSS}$, as $\textsc{adm}$ needs to be correctly recoverable. This proves the theorem. This concrete example is possible, as we only use required behavior.

Theorem 7 (Weak Blockwise Non-Interactive Public Accountability is Required for any Transform $\mathcal{T}$). *For any transformation algorithm $\mathcal{T}$, the utilized $\mathcal{SSS}$ must be weakly blockwise non-interactive publicly accountable to result in an unforgeable $\mathcal{RSS}$.*

Proof. Let $\mathcal{RSS}'$ be the resulting $\mathcal{RSS}$ from the given $\mathcal{SSS}$. Perform the following steps to show that the used $\mathcal{SSS}$ is not weakly blockwise non-interactive publicly accountable. In particular, let $\mathcal{A}$ be an adversary winning the unforgeability game, which is used by $\mathcal{B}$ to break the weak blockwise non-interactive public accountability of the used $\mathcal{SSS}$.

1. The challenger generates the two key pairs of the $\mathcal{SSS}$. It passes all keys but sk_{sig} to $\mathcal{B}$
2. $\mathcal{B}$ forwards all received keys to $\mathcal{A}$
3. $\mathcal{A}$ transforms the $\mathcal{SSS}$ into $\mathcal{RSS}'$ given the transform $\mathcal{T}$

4. Any calls to the signing oracle by $\mathcal{A}$ are answered genuinely by $\mathcal{B}$ using its own signing oracle
5. Eventually, $\mathcal{A}$ returns a tuple $(m, \sigma_{\mathcal{RSS}})$ to $\mathcal{B}$
6. If the resulting signature does not verify or does not win the unforgeability game, $\mathcal{A}$ and therefore also $\mathcal{B}$ abort
7. $\mathcal{B}$ outputs the underlying message-signature pair $(m', \sigma_{\mathcal{SSS}'})$

Following Fig. 3, $(m', \sigma_{\mathcal{SSS}'})$ breaks the weak blockwise non-interactive public accountability requirement of the $\mathcal{SSS}$, as there exists a block, which has not been signed by the signer, while the signer is accused by Detect. Moreover, the success probabilities are equal. The contrary, i.e., if the $\mathcal{SSS}$ used is not weakly blockwise non-interactive publicly accountable, the proof is similar. To achieve the correctness requirements, our accountability definition must hold blockwise.

Theorem 8 (No Unforgeable $\mathcal{RSS}$ can be Transformed into an $\mathcal{SSS}$). *There exists no transform $\mathcal{T}$, which converts an unforgeable $\mathcal{RSS}$ into an unforgeable $\mathcal{SSS}$.*

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

1. The challenger generates a key pair for an $\mathcal{RSS}$. It passes pk to $\mathcal{A}$.
2. $\mathcal{A}$ transforms $\mathcal{RSS}$ into $\mathcal{SSS}'$ given the transform $\mathcal{T}$
3. $\mathcal{A}$ calls the oracle $\mathcal{RSS}.\mathsf{Sign}$ with a message $m = (1, 2)$ and simulates $\mathcal{SSS}'.\mathsf{Sign}$ with $\textsc{adm} = (1)$
4. $\mathcal{A}$ calls $\mathcal{SSS}'.\mathsf{Sanit}$ with $\textsc{mod} = (1, a)$, $a \in_R \{0,1\}^\lambda$.
5. If the resulting signature does not verify, abort
6. Output the resulting signature $\sigma_{\mathcal{RSS}}$ of the underlying $\mathcal{RSS}$

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

B Proofs of Theorem 3 and 4

Th. 3: There exists an $\mathcal{SSS}$ which is private, but not strongly private.

Proof. We do so by modifying an arbitrary existing strongly private $\mathcal{SSS}$. Let $\mathcal{SSS} = (\mathsf{KGen}_{\mathrm{sig}}, \mathsf{KGen}_{\mathrm{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ be an arbitrary private $\mathcal{SSS}$. We alter the scheme as follows:

– $\mathsf{KGen}'_{\mathrm{sig}} := \mathsf{KGen}_{\mathrm{sig}}$
– $\mathsf{KGen}'_{\mathrm{san}} := \mathsf{KGen}_{\mathrm{san}}$, while an additional key pair for a IND-CCA2-secure encryption scheme ENC is generated.
– Sign' is the same as Sign, but it appends the encryption e of a digest of original message to the final signature, i.e., $\sigma' = (\sigma, e)$, where $e \leftarrow \mathsf{ENC}(pk_{\mathrm{san}}, \mathcal{H}(m))$ and $\mathcal{H}$ some cryptographic hash-function.
– Sanit' is the same as Sanit, while it first removes the encrypted digest from the signature, appending it to the resulting signature.

– Verify', Proof' and Judge' work the same as their original counterparts, but removing the trailing e from the signature before proceeding.

Clearly, a sanitizer holding the corresponding secret key for ENC, can distinguish between *messages* generated by the signer and the sanitizer using the information contained in the *signature* σ. Without sk_{san}, this information remains hidden due to the IND-CCA2 encryption.

Th. 4: Our Construction is Secure. We have to show that the resulting $\mathcal{RSS}$ is unforgeable and weakly private, but not strongly.

Proof. We prove each property on its own.

I) **Unforgeability.** Let $\mathcal{A}$ be an algorithm breaking the unforgeability of the resulting $\mathcal{RSS}$. We can then construct an algorithm $\mathcal{B}$ which breaks the weak blockwise non-interactive public accountability of the utilized $\mathcal{SSS}$. To do so, $\mathcal{B}$ simulates $\mathcal{A}$'s environment in the following way:
   1. $\mathcal{B}$ receives $pk_{\text{san}}, sk_{\text{san}}, pk_{\text{sig}}$ and forwards them to $\mathcal{A}$
   2. $\mathcal{B}$ forwards every query to its own signing oracle
   3. Eventually, $\mathcal{A}$ outputs a tuple (m^*, σ^*)
   4. If (m^*, σ^*) does not verify or is trivial, abort
   5. $\mathcal{B}$ outputs (m^*, σ^*)

   m cannot be derived from any queried message, with the exception of $m[i] = \square$ for any index i. Hence, $\exists i : m[i] \neq \square$, which has not been signed by the signer. The accepting verification requires that $\texttt{Sig} = \mathsf{Detect}(1^\lambda, m^*, \sigma^*, pk_{\text{sig}}, pk_{\text{san}})$. Therefore, (m^*, σ^*) breaks the weak blockwise non-interactive publicly accountability. The success probability of $\mathcal{B}$ equals the one of $\mathcal{A}$.

II) **Weak Privacy.** To show that our scheme is weakly private, we only need to show that an adversary $\mathcal{A}$ cannot derive information about the prior content of a contained block $m[i]$, as $\square$ is considered part of the resulting message m' and all other modifications result in a forgeable $\mathcal{RSS}$. Let $\mathcal{A}$ winning the weak privacy game. We can then construct an adversary $\mathcal{B}$ which breaks the strong privacy game in the following way:
   1. $\mathcal{B}$ receives $pk_{\text{san}}, sk_{\text{san}}, pk_{\text{sig}}$ and forwards them to $\mathcal{A}$
   2. $\mathcal{B}$ forwards every query to its own oracles
   3. Eventually, $\mathcal{A}$ outputs its guess b^*

   $\mathcal{B}$ outputs b^* as its own guess. The oracle requires that $\textsc{mod}_1(m_1) = \textsc{mod}(m_2)$, disregarding $\square$. Note, the messages are the same. Hence, the success probability of $\mathcal{B}$ is the same as $\mathcal{A}$'s. This proves the theorem.

III) **No Strong Privacy.** Due to the above, 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., $\square$, 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, $\ell_0 < \ell_1$, while m_0 is a prefix of m_1. Afterwards, it requests that $m_1[\ell_1]$ is redacted, i.e., $\textsc{mod}_1 = (\ell_1)$ and $\textsc{mod}_0 = ()$. Hence, if the oracle chooses $b = 0$, it will output $m_2 = m_0$ and for $b = 1$, $m_2 = (m_1, \square)$. Hence, the adversary wins the game, as $(m_1, \square) \neq m_0$.