Rethinking Privacy for Extended Sanitizable Signatures and a Black-Box Construction of Strongly Private Schemes*

David Derler and Daniel Slamanig

IAIK, Graz University of Technology, Austria
{david.derler|daniel.slamanig}@tugraz.at

Abstract. Sanitizable signatures, introduced by Ateniese et al. at ESORICS'05, allow to issue a signature on a message where certain predefined message blocks may later be changed (sanitized) by some dedicated party (the sanitizer) without invalidating the original signature. With sanitizable signatures, replacements for modifiable (admissible) message blocks can be chosen *arbitrarily* by the sanitizer. However, in various scenarios this makes sanitizers *too powerful*. To reduce the sanitizers power, Klonowski and Lauks at ICISC'06 proposed (among others) an extension that enables the signer to limit the allowed modifications per admissible block to a well defined set each. At CT-RSA'10 Canard and Jambert then extended the formal model of Brzuska et al. from PKC'09 to additionally include the aforementioned and other extensions. We, however, observe that the privacy guarantees of their model do not capture privacy in the sense of the original definition of sanitizable signatures. That is, if a scheme is private in this model it is not guaranteed that the sets of allowed modifications remain concealed. To this end, we review a stronger notion of privacy, i.e., (strong) unlinkability (defined by Brzuska et al. at EuroPKI'13), in this context. While unlinkability fixes this problem, no efficient unlinkable scheme supporting the aforementioned extensions exists and it seems to be hard to construct such schemes. As a remedy, in this paper, we propose a notion stronger than privacy, but weaker than unlinkability, which captures privacy in the original sense. Moreover, it allows to easily construct efficient schemes satisfying our notion from secure existing schemes in a black-box fashion.

1 Introduction

Digital signatures are an important cryptographic tool to assert the authenticity (source) and integrity of digital content. By virtue of these desired properties, every alteration of signed data necessarily yields an invalidation of the original signature. If one, however, considers handwritten signatures on paper documents, there are various scenarios where the handwritten signature is still

* The authors have been supported by EU Horizon 2020 through project PRISMACLOUD, grant agreement number 644962. This is the full version of a paper to appear at ProvSec 2015.

visible (source authentication is still given), but the document contains several blacked-out sections. These sections are not readable anymore and thus remain confidential. Examples for such sanitized documents are the public release of previously classified governmental documents, legal subpoenas for documents in court trials or documents for medical or biomedical research [1, 13, 15].

It is clear that conventional digital signatures can not be used as a means for source authentication in such scenarios for the obvious reason. A naive solution would be to issue a fresh signature on a sanitized version of the respective document. However, this is often not possible (e.g., the signing key has already expired or is not available) or it is even undesirable (e.g., due to time or cost constraints).

1.1 Background on Sanitizable Signatures

To realize a controlled and limited sanitization of digitally signed content without signer-interaction, various approaches to so called sanitizable signatures have been introduced and refined over the years. Today, there are essentially two flavors of sanitizable signatures. The first one focuses on removal (blacking-out) of designated parts not necessarily conducted by a designated party (could be everyone) and it covers redactable signatures [20], content-extraction signatures [29] and the sanitizable signatures in [23]. The second one focuses on replacement of designated parts conducted only by a designated party (the sanitizer) and covers sanitizable signatures as defined in [2] and follow up work [4–8, 26]. For a separation of these flavors we refer the reader to [22].

In addition to the motivating examples in the beginning, sanitizable signatures have shown to be a useful tool in various scenarios. Their applications include customizing authenticated multicast transmissions, database outsourcing (combating software piracy and unauthorized content distribution), remote integrity checking of outsourced data [14] and secure routing [2]. Moreover, they find applications in the context of public sector (open government) data [30], DRM licensing for digital content protection [11, 32], privacy protection in smart grids [24], privacy-aware management of audit-log data [19], health record disclosure [3] and anonymization [27], as well as identity management [28, 33]. On the more theoretical side, it has been shown how to build attribute-based anonymous credential systems from sanitizable signatures in a black-box fashion [12].

In this paper, we focus on sanitizable signatures in the vein of Ateniese et al. [2]. The basic idea behind such a scheme is that a message is split into fixed and modifiable (admissible) blocks, where each admissible block is replaced by a chameleon hash (a trapdoor collision resistant hash) of this block, and the concatenation of all blocks is then signed. A sanitizer being in possession of the trapdoor, can then change each admissible block arbitrarily by computing collisions. Such a sanitizable signature scheme needs to satisfy (1) *unforgeability*, which says that no one except the honest signer and sanitizer can create valid signatures and sanitizations respectively, (2) *immutability*, which says that a malicious sanitizer must not be able to modify any part of the message which has not been specified as admissible by the signer, (3) *privacy*, which says that all

sanitized information is unrecoverable for anyone except signer and sanitizer, (4) *transparency*, which says that signatures created by the signer or the sanitizer are indistinguishable, and (5) *accountability*, which requires that a malicious signer or sanitizer is not able to deny authorship. These security properties have later been rigorously defined in [4], where it is also shown that accountability implies unforgeability, transparency implies privacy[1] and all other properties are independent. Later, the property of (strong) unlinkability [6, 8] as an even stronger privacy property has been introduced. Additionally, other properties such as (blockwise) non-interactive public accountability [7] have been proposed and the model has also been extended to cover several signers and sanitizers simultaneously [10].

1.2 Motivation for this Work

With sanitizable signatures, admissible blocks can be replaced *arbitrarily* by the sanitizer. However, this often makes sanitizers *too powerful* and thus may limit their applicability in various scenarios severely. To reduce the sanitizers' power, Klonowski and Lauks [21] introduced several extensions for sanitizable signatures, which allow to limit the power of a sanitizer in several ways and thus eliminate the aforementioned concerns. In particular, they have introduced extensions (1) limiting the set of possible modifications for an admissible block (`LimitSet`), (2) forcing the sanitizer to make the same changes in logically linked admissible blocks (`EnforceModif`), (3) limiting the sanitizer to modify at most k out of n admissible blocks (`LimitNbModif`) and (4) forcing the sanitizer to construct less than ℓ versions of a message (`LimitNbSanit`). Later, Canard and Jambert [9] extended the security model of Brzuska et al. [4] to cover the aforementioned extensions (as [21] did not provide any model or proofs).

The `LimitSet` Extension. Although all of the aforementioned features improve the applicability of sanitizable signatures, we deem the `LimitSet` extension to be the generally most useful one (besides, it is the only extension that is related to the privacy property). Thus, in the remainder of this paper, we only consider the `LimitSet` extension and refer to schemes that implement this extension as *extended sanitizable signature schemes* (ESSS). In existing constructions, `LimitSet` is realized by using cryptographic accumulators, a primitive that allows to succinctly represent a set (as a so called accumulator) and to compute witnesses certifying membership for elements in the set. Basically, the set of admissible changes for such a block is accumulated and the admissible block is replaced by the respective accumulator. Loosely speaking, the signer initially provides an element together with the witness and sanitizing simply requires the sanitizer to exchange this element and the witness.

How to define Privacy? Recall that for sanitizable signatures without extensions, privacy means that it should not be possible to recover the original message

[1] We note that the implication of privacy by transparency [6] only holds in the proof-restricted case (cf. Section 3).

from a sanitized version. Now, what is the most reasonable definition for privacy given the `LimitSet` extension? It seems to be most natural to require that, given a (sanitized) signature, a `LimitSet` block does not leak any information about the remaining elements in the respective set (and thus no information about the original message). By carefully inspecting the security model for ESSS in [9], we, however, observe that their privacy definition *does not* capture this. In fact, an ESSS that reveals *all* elements of the sets corresponding to `LimitSet` blocks *will be private* in their model. One motivation for a weak definition of privacy in [9] might have been to preserve the implication from (proof-restricted) transparency (as in the original model from [4]). However, as it totally neglects any privacy guarantees for the `LimitSet` extension, a stronger privacy notion seems advantageous and often even required. In [6, 8] a stronger notion of privacy for sanitizable signatures—called (strong) unlinkability—has been introduced. This notion, when adapted to ESSS, indeed guarantees what we want to achieve. Yet, unlinkability induces a significant overhead for constructions supporting the `LimitSet` extension. As we will see later, the only unlinkable construction that supports the `LimitSet` extension [12] is rather inefficient and is only proven secure in a customized model which *does not* consider all security requirements of sanitizable signatures and thus does not represent an ESSS. In general, as we will show later, efficient unlinkable constructions of ESSS seem hard to achieve. Taking all together we conclude that, while the notion of privacy in [9] seems to be too weak, unlinkability seems to be too strong. Subsequently, we motivate why a stronger privacy notion (inbetween these two notions) that still allows to obtain efficient instantiations is however important for practical applications.

Motivating Applications. We consider use cases where it is required to limit the sanitizers abilities, while at the same time providing privacy with respect to verifiers. For instance, consider authenticity preserving workflows that span multiple enterprises. Using ESSS they can be modeled as illustrated in Figure 1, with a signer and a sanitizer per enterprise. Then, employees can—within some well defined boundaries—act (in the role of the sanitizer) on behalf of their company, while also being accountable for their actions. However, companies do not disclose sensitive business internals. As a concrete example for such a workflow,

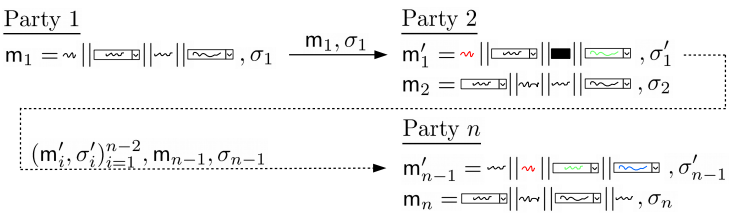

Fig. 1. Modeling a Workflow using ESSS

envision that a bank signs a document containing a `LimitSet` block with autho-

rized financial transactions for some company once every day. An employee of this company is then able to demonstrate the authorization of single transactions to subsequent enterprises via sanitization, while not being able to maliciously introduce new transactions. The company will definitely want that employees can be held accountable for revealing certain transactions and that transactions which were never revealed by sanitized versions of the orignal document remain concealed. Observe, that an ESSS being private according to [9] could reveal sensitive business internals upon signature verification (i.e., the unused transaction information). Another use case is the anonymization of (medical) data before publishing it, e.g., instead of removing the entire address information of some individual, one can replace the precise address with some larger region. To do so, one could define an admissible set with two elements being the precise address and the region. This would greatly help to automate the sanitization and to reduce errors, which, in turn, improves the quality of sanitized documents.[2] Likewise to the previous example, an ESSS which is private according to the definition in [9] would allow to reconstruct the precise address from a sanitized document.

1.3 Contribution

In this paper we take a closer look at the privacy definition for ESSS in [9] as well as the unlinkability definitions in [6, 8] when applied to the security model for ESSS. We conclude that these notions are either not strict enough to cover the requirements outlined in the previous section or too strict to obtain practical schemes. To this end, we introduce a stronger notion of privacy—denoted *strong privacy*—which explicitly considers privacy issues related to the `LimitSet` extension. More precisely, our strengthened notion guarantees that the sets of allowed modifications remain concealed, while still allowing efficient instantiations. We show that *privacy* is strictly weaker than *strong privacy* and that *unlinkability* is strictly stronger than *strong privacy*. Most importantly, we show that efficient and secure ESSS providing strong privacy can be constructed in a *black-box way* from *any* sanitizable signature scheme that is secure in the models of [4, 18]. We do so by proposing (1) a generic conversion of sanitizable signatures to ESSS which support the `LimitSet` extension and (2) showing that instantiating the `LimitSet` extension in this generic conversion with indistinguishable accumulators (as introduced in [16]) yields constructions that provide strong privacy.

2 Preliminaries and Notation

For the sake of compact notation, we often use the concatenation operator, e.g., $(a_i)_{i=1}^n || (b_i)_{i=1}^m := (a_1, \ldots, a_n, b_1, \ldots, b_m)$ and assume that concatenated sequences can later be uniquely decomposed (even when concatenating elements

[2] Such sets could be obtained and standardized by using concepts from k-anonymity [31] or t-plausibility [1] with the help of domain expert knowledge.

of different types and lengths). Let $x \xleftarrow{R} X$ denote the operation that picks an element x uniformly at random from a finite set X. A function $\epsilon : \mathbb{N} \to \mathbb{R}^+$ is called negligible if for all $c > 0$ there is a k_0 such that $\epsilon(k) < 1/k^c$ for all $k > k_0$. In the remainder of this paper, we use ϵ to denote such a negligible function.

2.1 (Indistinguishable) Accumulators

Accumulators allow to represent a finite set $\mathcal{X}$ of values as a single succinct accumulator $\mathsf{acc}_{\mathcal{X}}$. For each value $x \in \mathcal{X}$, one can efficiently obtain a membership witness wit_x that certifies the membership of x in $\mathsf{acc}_{\mathcal{X}}$, while this is infeasible for values $y \notin \mathcal{X}$ (collision freeness). Indistinguishable accumulators [16] additionally require that neither the accumulator nor corresponding witnesses leak information about the accumulated set. Subsequently, we use the basic model for static accumulators from [16] and note that in general a trusted setup is assumed (i.e., AGen is run by a TTP that discards the trapdoor $\mathsf{sk}_{\mathsf{acc}}$). However, if the party maintaining $\mathsf{acc}_{\mathcal{X}}$ is trusted, as it is the case within sanitizable signatures, using $\mathsf{sk}_{\mathsf{acc}}$ may be useful as it typically supports more efficient computations (the parameter $\mathsf{sk}_{\mathsf{acc}}^{\sim}$ denotes the optional trapdoor, i.e., using the trapdoor does not influence the output distributions of the algorithms and all algorithms also run without $\mathsf{sk}_{\mathsf{acc}}$).

Definition 1 (Accumulator [16]). *An accumulator is a tuple of PPT algorithms* (AGen, AEval, AWitCreate, AVerify) *which are defined as follows:*

$\mathsf{AGen}(1^\kappa, t)$**:** *This algorithm takes a security parameter κ and a parameter t. If $t \neq \infty$, then t is an upper bound for the number of accumulated elements. It returns a key pair* $(\mathsf{sk}_{\mathsf{acc}}, \mathsf{pk}_{\mathsf{acc}})$*, where* $\mathsf{sk}_{\mathsf{acc}} = \emptyset$ *if no trapdoor exists.*

$\mathsf{AEval}((\mathsf{sk}_{\mathsf{acc}}^{\sim}, \mathsf{pk}_{\mathsf{acc}}), \mathcal{X})$**:** *This (probabilistic)*[3] *algorithm takes a key pair* $(\mathsf{sk}_{\mathsf{acc}}^{\sim}, \mathsf{pk}_{\mathsf{acc}})$ *and a set $\mathcal{X}$ to be accumulated and returns an accumulator* $\mathsf{acc}_{\mathcal{X}}$ *together with some auxiliary information* aux*.*

$\mathsf{AWitCreate}((\mathsf{sk}_{\mathsf{acc}}^{\sim}, \mathsf{pk}_{\mathsf{acc}}), \mathsf{acc}_{\mathcal{X}}, \mathsf{aux}, x)$**:** *This algorithm takes a key pair* $(\mathsf{sk}_{\mathsf{acc}}^{\sim}, \mathsf{pk}_{\mathsf{acc}})$*, an accumulator* $\mathsf{acc}_{\mathcal{X}}$*, auxiliary information* aux *and a value x. It returns $\perp$, if $x \notin \mathcal{X}$, and a witness* wit_x *for x otherwise.*

$\mathsf{AVerify}(\mathsf{pk}_{\mathsf{acc}}, \mathsf{acc}_{\mathcal{X}}, \mathsf{wit}_x, x)$**:** *This algorithm takes a public key* $\mathsf{pk}_{\mathsf{acc}}$*, an accumulator* $\mathsf{acc}_{\mathcal{X}}$*, a witness* wit_x *and a value x. It returns* `true` *if* wit_x *is a witness for $x \in \mathcal{X}$ and* `false` *otherwise.*

A secure indistinguishable accumulator is correct, collision free and indistinguishable. We recall the definitions for collision freeness and indistinguishability below.[4]

[3] If AEval is probabilistic, the internally used randomness is denoted as r. AEval_r is used to make the randomness explicit.

[4] Note that, even though $\mathcal{A}$ can run AEval and AWitCreate itself, they are modeled as oracles to emphasize that $\mathcal{A}$ sees arbitrary accumulators and witnesses.

Definition 2 (Collision Freeness). *An accumulator is* collision-free*, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that:*

$$\mathsf{Pr}\left[\begin{array}{c}(\mathsf{sk}_{\mathsf{acc}},\mathsf{pk}_{\mathsf{acc}})\leftarrow\mathsf{AGen}(1^\kappa,t),\ \mathcal{O}\leftarrow\{\mathcal{O}^{\mathsf{E}(\cdot,\cdot,\cdot)},\mathcal{O}^{\mathsf{W}(\cdot,\cdot,\cdot,\cdot)}\},\\ (\mathsf{wit}_x^*,x^*,\mathcal{X}^*,r^*)\leftarrow\mathcal{A}^{\mathcal{O}}(\mathsf{pk}_{\mathsf{acc}}):\\ (\mathsf{AVerify}(\mathsf{pk}_{\mathsf{acc}},\mathsf{acc}^*,\mathsf{wit}_x^*,x^*)=\mathtt{true}\ \wedge\ x^*\notin\mathcal{X}^*)\end{array}\right]\leq\epsilon(\kappa),$$

where $\mathsf{acc}^*\leftarrow\mathsf{AEval}_{r^*}((\mathsf{sk}_{\mathsf{acc}},\mathsf{pk}_{\mathsf{acc}}),\mathcal{X}^*)$. Here, $\mathcal{O}^{\mathsf{E}}$ and $\mathcal{O}^{\mathsf{W}}$ represent the oracles for the algorithms AEval *and* AWitCreate*, respectively. In case of randomized accumulators the adversary outputs randomness r^*, which is omitted for deterministic accumulators. Likewise, the adversary can control the randomness r used by $\mathcal{O}^{\mathsf{E}}$ for randomized accumulators. Thus $\mathcal{O}^{\mathsf{E}}$ takes an additional input r.*

Definition 3 (Indistinguishability). *An accumulator is* indistinguishable*, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that:*

$$\mathsf{Pr}\left[\begin{array}{c}(\mathsf{sk}_{\mathsf{acc}},\mathsf{pk}_{\mathsf{acc}})\leftarrow\mathsf{AGen}(1^\kappa,t),\ b\xleftarrow{R}\{0,1\},\ (\mathcal{X}_0,\mathcal{X}_1,\\ \mathsf{state})\leftarrow\mathcal{A}(\mathsf{pk}_{\mathsf{acc}}),(\mathsf{acc}_{\mathcal{X}_b},\mathsf{aux})\leftarrow\mathsf{AEval}((\widetilde{\mathsf{sk}_{\mathsf{acc}}},\mathsf{pk}_{\mathsf{acc}}),\mathcal{X}_b),\\ \mathcal{O}\leftarrow\{\mathcal{O}^{\mathsf{E}(\cdot,\cdot,\cdot)},\mathcal{O}^{\mathsf{W}(\cdot,\cdot,\mathsf{aux},\cdot)}\},\ b^*\leftarrow\mathcal{A}^{\mathcal{O}}(\mathsf{pk}_{\mathsf{acc}},\mathsf{acc}_{\mathcal{X}_b},\mathsf{state}):\\ b=b^*\end{array}\right]\leq\frac{1}{2}+\epsilon(\kappa),$$

where $\mathcal{X}_0$ and $\mathcal{X}_1$ are two distinct subsets of the accumulation domain. Here, $\mathcal{O}^{\mathsf{E}}$ is defined as before, whereas $\mathcal{O}^{\mathsf{W}}$ is restricted to queries for values $x\in\mathcal{X}_0\cap\mathcal{X}_1$. Furthermore, the input parameter aux *for $\mathcal{O}^{\mathsf{W}}$ is kept up to date and is provided by the environment, since $\mathcal{A}$ could trivially distinguish using* aux*.*

It is obvious, that the notion of indistinguishability requires a randomized AEval algorithm. We stress that [16] also provide a variant of indistinguishability, which adds this non-determinism to accumulators with a deterministic AEval algorithm. To do so, an additional random value x_r from the accumulation domain is inserted into the accumulator upon AEval. This notion is called collision-freeness weakening (cfw) indistinguishability, since collision freeness only holds for $\mathcal{X}\cup\{x_r\}$ and not $\mathcal{X}$.

3 Formalizing Extended Sanitizable Signatures

In this section, we present a formal model for ESSS. Our model can thereby be seen as a rigorous formalization of the model for ESSS presented in [9]. Additionally, we include the suggestions from [18], i.e., additionally consider forgeries where one only tampers with ADM. We stress that, when omitting the extensions regarding `LimitSet` and ADM, it is equivalent to the model of [4], which is generally considered as the standard model for sanitizable signature schemes.

Definition 4 (Message). *A message $\mathsf{m}=(\mathsf{m}_i)_{i=1}^n$ is a sequence of n bitstrings (message blocks).*

Henceforth, we use ℓ_i to refer to the (maximum) length of message block m_i and assume an encoding that allows to derive $(\ell_i)_{i=1}^n$ from m.

Definition 5 (Admissible Modifications). *Admissible modifications* ADM *with respect to a message* $\mathsf{m} = (\mathsf{m}_i)_{i=1}^n$ *are represented as a sequence* $\mathsf{ADM} = (\mathsf{B}_i)_{i=1}^n$*, with* $\mathsf{B}_i \in \{\mathtt{fix}, \mathtt{var}, \mathtt{lim}\}$*.*

Here $\mathsf{B}_i = \mathtt{fix}$ indicates that no changes are allowed, $\mathsf{B}_i = \mathtt{var}$ indicates that arbitrary replacements are allowed, and $\mathsf{B}_i = \mathtt{lim}$ indicates that the replacements are limited to a predefined set (`LimitSet`).

Definition 6 (Set Limitations). *Set limitations* V *with respect to a message* $\mathsf{m} = (\mathsf{m}_i)_{i=1}^n$ *and admissible modifications* $\mathsf{ADM} = (\mathsf{B}_i)_{i=1}^n$ *are represented by a set* $\mathsf{V} = \{(i, \mathsf{M}_i) : \mathsf{B}_i = \mathtt{lim} \ \wedge \ \mathsf{M}_i \subset \bigcup_{j=0}^{\ell_i} \{0,1\}^j\}$*.*

We use $\mathsf{m}' \overset{(\mathsf{ADM},\mathsf{V})}{\preceq} \mathsf{m}$ to denote that m' can be derived from m under ADM and V.

Definition 7 (Witnesses). *Witnesses* $\mathcal{W} = \{(i, \mathcal{W}_i)\}_{i=1}^t$*, with* $\mathcal{W}_i = \{(m_{i_1}, \mathsf{wit}_{i_1}), \ldots, (m_{i_k}, \mathsf{wit}_{i_k})\}$*, are derived from set limitations* $\mathsf{V} = \{(i, \mathsf{M}_i)\}_{i=1}^t$*, with* $\mathsf{M}_i = \{m_{i_1}, \ldots, m_{i_k}\}$*. Thereby,* wit_{i_j} *attests that its corresponding message block* m_{i_j} *is contained in the set* M_i*.*

With $\mathcal{V} \overset{(\mathsf{m},\mathsf{ADM})}{\longleftarrow} \mathcal{W}$, we denote the extraction of the set of witnesses $\mathcal{V}$ corresponding to a message m from the set $\mathcal{W}$.

Definition 8 (Modification Instructions). *Modification instructions* MOD*, with respect to a message* $\mathsf{m} = (\mathsf{m}_i)_{i=1}^n$*, admissible modifications* ADM *and set limitations* V *are represented by a set* $\mathsf{MOD} = \{(i, \mathsf{m}'_i)\}_{i=1}^t$ *with* $t \leq n$*, where* i *refers to the position of the message block in* m*, and* m'_i *is the new content for message block* m_i*.*

With $\mathsf{MOD} \preceq (\mathsf{ADM}, \mathsf{V})$, we denote that the modification instructions in MOD are compatible with ADM and V. Furthermore, with $(\mathsf{m}_0, \mathsf{MOD}_0, \mathsf{ADM}, \mathsf{V}) \equiv (\mathsf{m}_1, \mathsf{MOD}_1, \mathsf{ADM}, \mathsf{V})$, we denote that after applying the changes in MOD_0 and MOD_1 to m_0 and m_1 respectively, the resulting messages m'_0 and m'_1 are identical.

3.1 The Model

An ESSS is a tuple of PPT algorithms $(\mathsf{KeyGen}_{\mathsf{sig}}, \mathsf{KeyGen}_{\mathsf{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ which are defined as follows:

$\mathsf{KeyGen}_{\mathsf{sig}}(1^\kappa)$**:** This algorithm takes as input a security parameter κ and outputs a keypair $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}})$ for the signer.

$\mathsf{KeyGen}_{\mathsf{san}}(1^\kappa)$**:** This algorithm takes as input a security parameter κ and outputs a keypair $(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}})$ for the sanitizer.

$\mathsf{Sign}(\mathsf{m}, \mathsf{ADM}, \mathsf{V}, (\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), \mathsf{pk}_{\mathsf{san}})$**:** This algorithm takes as input a message m, corresponding admissible modifications ADM and set limitations V, as well as the keypair $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}})$ of the signer and the verification key $\mathsf{pk}_{\mathsf{san}}$ of the sanitizer. It computes the set $\mathcal{W}$ from V, obtains $\mathcal{V} \overset{(\mathsf{m},\mathsf{ADM})}{\longleftarrow} \mathcal{W}$ and outputs a signature $\sigma = (\hat{\sigma}, \mathcal{V})$ together with some auxiliary sanitization information

$\mathsf{san} = (\mathsf{aux}, \mathcal{W})$.[5] In case of an error, $\perp$ is returned. As in [4], we assume that ADM can be recovered from a signature σ.

$\mathsf{Sanit}((\mathsf{m},\sigma), \mathsf{MOD}, \mathsf{san}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}})$**:** This algorithm takes as input a valid message-signature pair (m,σ), modification instructions MOD, some auxiliary sanitization information san and the verification key $\mathsf{pk}_{\mathsf{sig}}$ of the signer and the signing key $\mathsf{sk}_{\mathsf{san}}$ of the sanitizer. It modifies m and σ according to MOD and outputs an updated message-signature pair (m', σ') and $\perp$ if $\mathsf{m}' \not\preceq^{(\mathsf{ADM},\mathsf{V})} \mathsf{m}$. We assume that V can be reconstructed from san.

$\mathsf{Verify}((\mathsf{m},\sigma), \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$**:** This algorithm takes as input a message-signature pair (m, σ) and the public verification keys of the signer $\mathsf{pk}_{\mathsf{sig}}$ and the sanitizer $\mathsf{pk}_{\mathsf{san}}$. It returns `true` if σ is a valid signature on m under $\mathsf{pk}_{\mathsf{sig}}$ and $\mathsf{pk}_{\mathsf{san}}$, and `false` otherwise.

$\mathsf{Proof}((\mathsf{m},\sigma), \{(\mathsf{m}_j,\sigma_j)\}_{j=1}^{q}, (\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), \mathsf{pk}_{\mathsf{san}})$**:** This algorithm takes as input a message-signature pair (m,σ), q message-signature pairs $\{(\mathsf{m}_j,\sigma_j)\}_{j=1}^{q}$ created by the signer, the keypair $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}})$ of the signer and the public key $\mathsf{pk}_{\mathsf{san}}$ of the sanitizer and outputs a proof π.

$\mathsf{Judge}((\mathsf{m},\sigma), \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \pi)$**:** This algorithm takes as input a message-signature pair (m,σ), the verification keys of the signer $\mathsf{pk}_{\mathsf{sig}}$ and the sanitizer $\mathsf{pk}_{\mathsf{san}}$ and a proof π. It outputs a decision $d \in \{\mathtt{sig}, \mathtt{san}\}$, indicating whether the signature has been produced by the signer or the sanitizer.

3.2 Security Properties

For security, an ESSS is required to fulfill the following properties.

Definition 9 (Correctness). *An* ESSS *is correct, if*

$$
\begin{array}{l}
\forall\kappa, \forall \mathsf{m}, \forall \mathsf{ADM}, \forall \mathsf{V}, \forall \mathsf{MOD} \preceq (\mathsf{ADM}, \mathsf{V}), \\
\forall(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(1^\kappa), \forall(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(1^\kappa), \\
\forall(\sigma, \mathsf{san}) \leftarrow \mathsf{Sign}(\mathsf{m}, \mathsf{ADM}, \mathsf{V}, (\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), \mathsf{pk}_{\mathsf{san}}), \\
\forall(\mathsf{m}', \sigma') \leftarrow \mathsf{Sanit}((\mathsf{m},\sigma), \mathsf{MOD}, \mathsf{san}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}}), \\
\forall\{(\mathsf{m}_1, \mathsf{ADM}_1, \mathsf{V}_1), \ldots, (\mathsf{m}_q, \mathsf{ADM}_q, \mathsf{V}_q)\} \supseteq (\mathsf{m}, \mathsf{ADM}, \mathsf{V}) : \\
\mathsf{Verify}((\mathsf{m},\sigma), \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) = \mathtt{true} \ \wedge \ \mathsf{Verify}((\mathsf{m}', \sigma'), \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) = \mathtt{true} \ \wedge \\
\big(((\sigma_j, \cdot) \leftarrow \mathsf{Sign}(\mathsf{m}_j, \mathsf{ADM}_j, \mathsf{V}_j, (\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), \mathsf{pk}_{\mathsf{san}}))_{j=1}^{q} \ \wedge \ \pi \leftarrow \mathsf{Proof}((\mathsf{m}', \sigma'), \\
\{(\mathsf{m}_j, \sigma_j)\}_{j=1}^{q}, (\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), \mathsf{pk}_{\mathsf{san}}) \ \wedge \ \mathsf{Judge}((\mathsf{m}', \sigma'), \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \pi) = \mathtt{san}\big).
\end{array}
$$

Definition 10 (Unforgeability). *An* ESSS *is unforgeable, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that:*

$$
\mathsf{Pr}\left[\begin{array}{c}
(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(1^\kappa), (\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(1^\kappa), \\
\mathcal{O} \leftarrow \{\mathcal{O}^{\mathsf{Sign}}(\cdot,\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}),\cdot), \mathcal{O}^{\mathsf{Sanit}}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{san}}), \\
\mathcal{O}^{\mathsf{Proof}}(\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}),\cdot)\}, (\mathsf{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) \ : \\
\mathsf{Verify}(\mathsf{m}^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) = \mathtt{true} \ \wedge \\
(\mathsf{m}^*, \mathsf{ADM}^*, \mathsf{V}^*, \mathsf{pk}_{\mathsf{san}}) \notin L^{\mathsf{Sign}} \ \wedge \ ((\mathsf{m}^*, \sigma^*), \mathsf{ADM}^*, \mathsf{pk}_{\mathsf{sig}}) \notin L^{\mathsf{Sanit}}
\end{array}\right] \leq \epsilon(\kappa),
$$

[5] While san is not required for plain sanitizable signature schemes, ESSS additionally return san to pass auxiliary information, which is only relevant for the sanitizer.

where $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{O}^{\mathsf{Sanit}}$ *and* $\mathcal{O}^{\mathsf{Proof}}$ *simulate the* Sign, Sanit *and* Proof *algorithms, respectively. The environment keeps track of the queries to* $\mathcal{O}^{\mathsf{Sign}}$ *using* L^{Sign}. *Furthermore, it maintains a list* L^{Sanit} *containing the answers of* $\mathcal{O}^{\mathsf{Sanit}}$ *extended with* $\mathsf{pk}_{\mathsf{sig}}$ *and* ADM *from the respective oracle query.*

Definition 11 (Immutability). *An* ESSS *is immutable, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\epsilon(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{c}(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}})\leftarrow\mathsf{KeyGen}_{\mathsf{sig}}(1^\kappa),\mathcal{O}\leftarrow\{\mathcal{O}^{\mathsf{Sign}}(\cdot,\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),\cdot),\\ \mathcal{O}^{\mathsf{Proof}}(\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),\cdot)\},(\mathsf{pk}^*_{\mathsf{san}},\mathsf{m}^*,\sigma^*)\leftarrow\mathcal{A}^{\mathcal{O}}(\mathsf{pk}_{\mathsf{sig}})\ :\\ \mathsf{Verify}(\mathsf{m}^*,\sigma^*,\mathsf{pk}_{\mathsf{sig}},\mathsf{pk}^*_{\mathsf{san}})=\mathtt{true}\ \wedge\ \Big((\cdot,\cdot,\cdot,\mathsf{pk}^*_{\mathsf{san}})\notin L^{\mathsf{Sign}}\ \vee\\ \nexists\mathsf{m}^{*\,(\mathsf{ADM}^*,\mathsf{V}^*)}\!\!\preceq\mathsf{m}:(\mathsf{m},\mathsf{ADM}^*,\mathsf{V}^*,\mathsf{pk}^*_{\mathsf{san}})\in L^{\mathsf{Sign}}\Big)\end{array}\right]\leq\epsilon(\kappa),$$

where the oracles and the environment variables are as in Definition 10.

Definition 12 (Privacy). *An* ESSS *is private, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\epsilon(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{c}(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}})\leftarrow\mathsf{KeyGen}_{\mathsf{sig}}(1^\kappa),(\mathsf{sk}_{\mathsf{san}},\mathsf{pk}_{\mathsf{san}})\leftarrow\mathsf{KeyGen}_{\mathsf{san}}(1^\kappa),\\ b\xleftarrow{R}\{0,1\},\mathcal{O}\leftarrow\{\mathcal{O}^{\mathsf{Sign}}(\cdot,\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),\cdot),\\ \mathcal{O}^{\mathsf{Sanit}}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{san}}),\mathcal{O}^{\mathsf{Proof}}(\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),\cdot),\mathcal{O}^{\mathsf{LoRSanit}}(\cdot,\cdot,\cdot,\\ (\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),(\mathsf{sk}_{\mathsf{san}},\mathsf{pk}_{\mathsf{san}}),b)\},b^*\leftarrow\mathcal{A}^{\mathcal{O}}(\mathsf{pk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{san}})\ :\ b=b^*\end{array}\right]\leq\frac{1}{2}+\epsilon(\kappa),$$

where $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{O}^{\mathsf{Sanit}}$ *and* $\mathcal{O}^{\mathsf{Proof}}$ *are as in Definition 10.* $\mathcal{O}^{\mathsf{LoRSanit}}$ *is defined as follows:*

$\mathcal{O}^{\mathsf{LoRSanit}}((\mathsf{m}_0,\mathsf{MOD}_0),(\mathsf{m}_1,\mathsf{MOD}_1),\mathsf{ADM},(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),(\mathsf{sk}_{\mathsf{san}},\mathsf{pk}_{\mathsf{san}}),b)$:

1: *Randomly choose* V *(compatible with* MOD_0 *and* MOD_1*).*
2: *If* $\mathsf{MOD}_0\not\preceq(\mathsf{ADM},\mathsf{V})\ \vee\ \mathsf{MOD}_1\not\preceq(\mathsf{ADM},\mathsf{V})$*, return* $\perp$.
3: *If* $(\mathsf{m}_0,\mathsf{MOD}_0,\mathsf{ADM},\mathsf{V})\not\equiv(\mathsf{m}_1,\mathsf{MOD}_1,\mathsf{ADM},\mathsf{V})$*, return* $\perp$.
4: *Compute* $(\sigma_b,\mathsf{san}_b)\leftarrow\mathsf{Sign}(\mathsf{m}_b,\ \mathsf{ADM},\mathsf{V},\ (\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),\ \mathsf{pk}_{\mathsf{san}})$.
5: *Return* $(\mathsf{m}'_b,\sigma'_b)\leftarrow\mathsf{Sanit}((\mathsf{m}_b,\sigma_b),\ \mathsf{MOD}_b,\mathsf{san}_b,\mathsf{pk}_{\mathsf{sig}},\mathsf{sk}_{\mathsf{san}})$.

Observe that since V is internally chosen (and, thus, independent of the bit b) in $\mathcal{O}^{\mathsf{LoRSanit}}$, privacy holds independent of the adversaries capability to reconstruct the set limitations. Clearly, this *contradicts* a definition of privacy in a sense that sanitized signatures do not reveal the original message.

Definition 13 (Transparency). *An* ESSS *is transparent, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\epsilon(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{c}(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}})\leftarrow\mathsf{KeyGen}_{\mathsf{sig}}(1^\kappa),(\mathsf{sk}_{\mathsf{san}},\mathsf{pk}_{\mathsf{san}})\leftarrow\mathsf{KeyGen}_{\mathsf{san}}(1^\kappa),\\ b\xleftarrow{R}\{0,1\},\mathcal{O}\leftarrow\{\mathcal{O}^{\mathsf{Sign}}(\cdot,\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),\cdot),\mathcal{O}^{\mathsf{Sanit}}(\cdot,\cdot,\cdot,\cdot,\\ \mathsf{sk}_{\mathsf{san}}),\mathcal{O}^{\mathsf{Proof}}(\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),\cdot),\mathcal{O}^{\mathsf{Sanit/Sign}}(\cdot,\cdot,\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),\\ (\mathsf{sk}_{\mathsf{san}},\mathsf{pk}_{\mathsf{san}}),b)\},b^*\leftarrow\mathcal{A}^{\mathcal{O}}(\mathsf{pk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{san}})\ :\ b=b^*\end{array}\right]\leq\frac{1}{2}+\epsilon(\kappa),$$

where $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{O}^{\mathsf{Sanit}}$ *and* $\mathcal{O}^{\mathsf{Proof}}$ *are as in Definition 10. In addition,* $\mathcal{O}^{\mathsf{Proof}}$ *does not respond to queries for messages-signature pairs obtained using* $\mathcal{O}^{\mathsf{Sanit/Sign}}$. $\mathcal{O}^{\mathsf{Sanit/Sign}}$ *is defined as follows:*

$\mathcal{O}^{\mathsf{Sanit/Sign}}(\mathsf{m}, \mathsf{ADM}, \mathsf{V}, \mathsf{MOD}, (\mathsf{sk_{sig}}, \mathsf{pk_{sig}}), (\mathsf{sk_{san}}, \mathsf{pk_{san}}), b)$**:**

1: *If* $\mathsf{MOD} \not\preceq (\mathsf{ADM}, \mathsf{V})$, *return* $\perp$.
2: *Compute* $(\sigma, \mathsf{san}) \leftarrow \mathsf{Sign}(\mathsf{m}, \mathsf{ADM}, \mathsf{V}, (\mathsf{sk_{sig}}, \mathsf{pk_{sig}}), \mathsf{pk_{san}})$.
3: *Compute* $(\mathsf{m}', \sigma_0) \leftarrow \mathsf{Sanit}((\mathsf{m}, \sigma), \mathsf{MOD}, \mathsf{san}, \mathsf{pk_{sig}}, \mathsf{sk_{san}})$.
4: *Compute* $(\sigma_1, \mathsf{san}) \leftarrow \mathsf{Sign}(\mathsf{m}', \mathsf{ADM}, \mathsf{V}, (\mathsf{sk_{sig}}, \mathsf{pk_{sig}}), \mathsf{pk_{san}})$.
5: *Return* (m', σ_b).

Proof-restricted transparency [6]: $\mathcal{O}^{\mathsf{Proof}}$ does not answer queries for messages returned by $\mathcal{O}^{\mathsf{Sanit/Sign}}$. In the proof for the implication of privacy by transparency [4], $\mathcal{O}^{\mathsf{Sanit/Sign}}$ is used to simulate the $\mathcal{O}^{\mathsf{LoRSanit}}$ queries. Thus, note that the implication only holds if the privacy-adversary is restricted to $\mathcal{O}^{\mathsf{Proof}}$ queries for messages which do not originate from $\mathcal{O}^{\mathsf{LoRSanit}}$. To additionally rule out even stronger adversaries against privacy, i.e., such that privacy also holds after seeing proofs for the messages in question, one would need to prove privacy directly.

Definition 14 (Sanitizer-Accountability). *An* ESSS *is sanitizer-accountable, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\epsilon(\cdot)$ *such that:*

$$\mathsf{Pr}\left[\begin{array}{c}(\mathsf{sk_{sig}}, \mathsf{pk_{sig}}) \leftarrow \mathsf{KeyGen_{sig}}(1^\kappa), \mathcal{O} \leftarrow \{\mathcal{O}^{\mathsf{Sign}}(\cdot,\cdot,\cdot,(\mathsf{sk_{sig}}, \mathsf{pk_{sig}}),\cdot),\\ \mathcal{O}^{\mathsf{Proof}}(\cdot,\cdot,(\mathsf{sk_{sig}}, \mathsf{pk_{sig}}),\cdot)\}, (\mathsf{pk^*_{san}}, \mathsf{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk_{sig}}),\\ \pi \leftarrow \mathsf{Proof}((\mathsf{m}^*, \sigma^*), \mathtt{SIG}, (\mathsf{sk_{sig}}, \mathsf{pk_{sig}}), \mathsf{pk^*_{san}}) : \mathsf{Verify}(\mathsf{m}^*, \sigma^*, \mathsf{pk_{sig}},\\ \mathsf{pk^*_{san}}) = \mathtt{true} \ \wedge \ (\mathsf{m}^*, \mathsf{ADM}^*, \mathsf{V}^*, \mathsf{pk^*_{san}}) \notin L^{\mathsf{Sign}} \ \wedge\\ \mathsf{Judge}((\mathsf{m}^*, \sigma^*), \mathsf{pk_{sig}}, \mathsf{pk^*_{san}}, \pi) = \mathsf{sig}\end{array}\right] \leq \epsilon(\kappa),$$

where the oracles are as in Definition 10. The environment maintains a list $\mathtt{SIG}$*, containing all message-signature tuples obtained from* $\mathcal{O}^{\mathsf{Sign}}$*.*

Definition 15 (Signer-Accountability). *An* ESSS *is signer-accountable, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\epsilon(\cdot)$ *such that:*

$$\mathsf{Pr}\left[\begin{array}{c}(\mathsf{sk_{san}}, \mathsf{pk_{san}}) \leftarrow \mathsf{KeyGen_{san}}(1^\kappa), \mathcal{O} \leftarrow \{\mathcal{O}^{\mathsf{Sanit}}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk_{san}})\},\\ (\mathsf{pk^*_{sig}}, \mathsf{m}^*, \sigma^*, \pi^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk_{san}}) : \mathsf{Verify}(\mathsf{m}^*, \sigma^*, \mathsf{pk^*_{sig}},\\ \mathsf{pk_{san}}) = \mathtt{true} \ \wedge \ ((\mathsf{m}^*, \sigma^*), \mathsf{ADM}^*, \mathsf{pk^*_{sig}}) \notin L^{\mathsf{Sanit}} \ \wedge\\ \mathsf{Judge}((\mathsf{m}^*, \sigma^*), \mathsf{pk^*_{sig}}, \mathsf{pk_{san}}, \pi) = \mathsf{san} \ \wedge\end{array}\right] \leq \epsilon(\kappa),$$

where $\mathcal{O}^{\mathsf{Sanit}}$ *as well as* L^{Sanit} *are as in Definition 10.*

4 Rethinking Privacy for ESSS

In the following, we consider alternatives to the standard privacy property, i.e., (strong) unlinkability, and finally come up with a notion denoted as strong privacy which captures privacy for ESSS in the original sense of sanitizable signatures.

4.1 Revisiting Unlinkability

The notion of unlinkability for sanitizable signatures has been introduced in [6] as a stronger notion of privacy (which implies the usual privacy property). In [8],

an even stronger notion, i.e., strong unlinkability, has been proposed. It requires that unlinkability must even hold for signers. The notions defined in [6, 8] can easily be adapted to the model for ESSS and we do so below.

Definition 16 (Unlinkability). *An* ESSS *is unlinkable, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that:*

$$\mathsf{Pr}\left[\begin{array}{c}(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}})\leftarrow\mathsf{KeyGen}_{\mathsf{sig}}(1^\kappa),(\mathsf{sk}_{\mathsf{san}},\mathsf{pk}_{\mathsf{san}})\leftarrow\mathsf{KeyGen}_{\mathsf{san}}(1^\kappa),\\ b\xleftarrow{R}\{0,1\},\mathcal{O}\leftarrow\{\mathcal{O}^{\mathsf{Sign}}(\cdot,\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),\cdot),\mathcal{O}^{\mathsf{Sanit}}(\cdot,\cdot,\cdot,\cdot,\\ \mathsf{sk}_{\mathsf{san}}),\mathcal{O}^{\mathsf{Proof}}(\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),\cdot),\mathcal{O}^{\mathsf{LoRSanit}}(\cdot,\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),\\ (\mathsf{sk}_{\mathsf{san}},\mathsf{pk}_{\mathsf{san}}),b)\},b^*\leftarrow\mathcal{A}^{\mathcal{O}}(\mathsf{pk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{san}})\ :\ b=b^*\end{array}\right]\leq\frac{1}{2}+\epsilon(\kappa),$$

where $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{O}^{\mathsf{Sanit}}$ and $\mathcal{O}^{\mathsf{Proof}}$ are as in Definition 10 and $\mathcal{O}^{\mathsf{LoRSanit}}$ operates as follows:

$\mathcal{O}^{\mathsf{LoRSanit}}((\mathsf{m}_0,\mathsf{MOD}_0,\mathsf{san}_0,\sigma_0),(\mathsf{m}_1,\mathsf{MOD}_1,\mathsf{san}_1,\sigma_1),\mathsf{ADM},(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}}),$
$(\mathsf{sk}_{\mathsf{san}},\mathsf{pk}_{\mathsf{san}}),b)$*:*
1: *If* $\mathsf{MOD}_0\not\preceq(\mathsf{ADM},\mathsf{V}_0)\ \vee\ \mathsf{MOD}_1\not\preceq(\mathsf{ADM},\mathsf{V}_1)$*, return* $\perp$.
2: *If* $(\mathsf{m}_0,\mathsf{MOD}_0,\mathsf{ADM},\mathsf{V}_0)\not\equiv(\mathsf{m}_1,\mathsf{MOD}_1,\mathsf{ADM},\mathsf{V}_1)$*, return* $\perp$.
3: *If for any* $i\in\{0,1\}$, $\mathsf{Verify}((\mathsf{m}_i,\sigma_i),\mathsf{pk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{san}})=\mathtt{false}$*, return* $\perp$
4: *Return* $(\mathsf{m}'_b,\sigma'_b)\leftarrow\mathsf{Sanit}((\mathsf{m}_b,\sigma_b),\mathsf{MOD}_b,\mathsf{san}_b,\mathsf{pk}_{\mathsf{sig}},\mathsf{sk}_{\mathsf{san}})$.

Note that V_0 and V_1 can be reconstructed from san_0 and san_1, respectively. Furthermore, note that for answers from the oracle $\mathcal{O}^{\mathsf{LoRSanit}}$, the oracle $\mathcal{O}^{\mathsf{Sanit}}$ is restricted to queries for modifications which are covered by both set limitations V_0 and V_1, which were initially submitted to $\mathcal{O}^{\mathsf{LoRSanit}}$.

Definition 17 (Strong Unlinkability). *An* ESSS *is strongly unlinkable, if for all PPT adversaries $\mathcal{A}$ there is a negligible function $\epsilon(\cdot)$ such that:*

$$\mathsf{Pr}\left[\begin{array}{c}(\mathsf{sk}_{\mathsf{san}},\mathsf{pk}_{\mathsf{san}})\leftarrow\mathsf{KeyGen}_{\mathsf{san}}(1^\kappa),b\xleftarrow{R}\{0,1\},\\ \mathcal{O}\leftarrow\{\mathcal{O}^{\mathsf{Sanit}}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{san}}),\mathcal{O}^{\mathsf{LoRSanit}}(\cdot,\cdot,\cdot,\cdot,(\mathsf{sk}_{\mathsf{san}},\mathsf{pk}_{\mathsf{san}}),b)\},\\ b^*\leftarrow\mathcal{A}^{\mathcal{O}}(\mathsf{pk}_{\mathsf{san}})\ :\ b=b^*\end{array}\right]\leq\frac{1}{2}+\epsilon(\kappa),$$

where the oracles are as in Definition 16, except that $\mathcal{A}$ controls $(\mathsf{sk}_{\mathsf{sig}},\mathsf{pk}_{\mathsf{sig}})$.

While (strong) unlinkability covers privacy for the `LimitSet` extension in the original sense of privacy[6], it seems very hard to construct efficient (strongly) unlinkable schemes that support the `LimitSet` extension. Unfortunately, it is not possible to simply extend existing (strongly) unlinkable constructions [6, 8, 17] by the `LimitSet` extension. To illustrate why, we revisit the design principle of such schemes. Here, upon Sign, the signer issues two signatures. The first signature, σ_{FIX}, only covers the fixed message blocks and the public key of the sanitizer, whereas the second signature, σ_{FULL}, covers the whole message together with the public key of the signer (and the public key of the sanitizer [8]). Upon Sanit, the sanitizer simply issues a new signature σ_{FULL}, whereas the

[6] Note, that the ability to reconstruct the set limitations for σ'_b obtained via $\mathcal{O}^{\mathsf{LoRSanit}}$ would imply a trivial distinguisher for the unlinkability game.

signature σ_{FIX} remains unchanged. Finally, upon Verify, one verifies whether σ_{FIX} is valid under $\mathsf{pk}_{\mathsf{sig}}$ and σ_{FULL} is either valid under $\mathsf{pk}_{\mathsf{sig}}$ or $\mathsf{pk}_{\mathsf{san}}$ for a given message m and ADM. Thereby, the signature scheme used for σ_{FIX} is a deterministic signature scheme, while the scheme used for σ_{FULL} can either also be a deterministic signature scheme [8], a group/ring signature scheme [6], or a signature scheme with rerandomizable keys [17].

When extending these schemes to also support the `LimitSet` extension, it is clear that the set limitations need to be fixed by the signer and must not be modifiable by the sanitizer. One simple way to realize the `LimitSet` extension would be to additionally include some unique encoding of the limitations V in σ_{FIX} and check whether the message is consistent with the defined limitations upon Verify. Obviously, this extension does not influence unforgeability and immutability and the scheme is still (publicly) accountable. Furthermore also privacy holds, since the set limitations which are included in the challenge tuple in the privacy game are randomly chosen inside the $\mathcal{O}^{\mathsf{LoRSanit}}$ oracle. However, unlinkability can not hold for the following reason: When querying the oracle $\mathcal{O}^{\mathsf{LoRSanit}}$ in the unlinkability game, the adversary can choose set limitations V_0 and V_1 such that $\mathsf{MOD}_0 \preceq (\mathsf{ADM}, \mathsf{V}_0)$, $\mathsf{MOD}_1 \preceq (\mathsf{ADM}, \mathsf{V}_1)$ and $(\mathsf{m}_0, \mathsf{MOD}_0, \mathsf{ADM}, \mathsf{V}_0) \equiv (\mathsf{m}_1, \mathsf{MOD}_1, \mathsf{ADM}, \mathsf{V}_1)$, but $\mathsf{V}_0 \neq \mathsf{V}_1$. For the corresponding signatures $\sigma_0 = (\sigma_{FIX_0}, \sigma_{FULL_0})$, $\sigma_1 = (\sigma_{FIX_1}, \sigma_{FULL_1})$ submitted to the oracle, this means that $\sigma_{FIX_0} \neq \sigma_{FIX_1}$ which yields a trivial distinguisher for the unlinkability game.

As an alternative, one may think of separately signing each message contained in the limited sets (using a deterministic signature scheme), where only the signatures corresponding to the chosen messages are revealed. However, to prevent forgeries where message blocks are re-used in other signatures (i.e., mix-and-match like attacks [5]), it would be required to also include some message-specific identifier in each signature. Again, it is easy to see that this would provide a trivial distinguisher for the (strong) unlinkability game.

Clearly, the requirement that the limited sets are fixed by the signer and cannot be modified later is not only specific to the aforementioned constructions, but is inherent to all constructions of such schemes. To circumvent the aforementioned issues, one could make use of more sophisticated primitives, which, however, come at the cost of significant computational overhead and complexity of the scheme. This is confirmed by the only known unlinkable construction supporting `LimitSet` [12]. It is computationally very expensive due to a high number of bilinear map applications and the use of non-interactive zero-knowledge proofs of knowledge in the computationally expensive target group of the bilinear map. Moreover, it is proven secure only in a model which does not consider all security requirements of sanitizable signatures (as it is tailored to their black-box construction of anonymous credentials) and thus does not represent an ESSS.

4.2 A Strengthened Notion for Privacy

Surprisingly, our requirement that the set limitations remain concealed can be met by a simple extension of the conventional privacy property. We call the

extended property *strong privacy.*[7] As we will see, this modification allows to obtain efficient implementations from secure existing ones in a black-box fashion. We modify the privacy game such that the set limitations in $\mathcal{O}^{\mathsf{LoRSanit}}$ can be submitted per message, i.e., $\mathcal{O}^{\mathsf{LoRSanit}}$ takes $(\mathsf{m}_0, \mathsf{MOD}_0, \mathsf{V}_0), (\mathsf{m}_1, \mathsf{MOD}_1, \mathsf{V}_1)$, ADM. This means that V_0 and V_1 can be different and only need an overlap such that after applying MOD_0 and MOD_1 the messages m'_0 and m'_1 are identical. More formally, the game is defined as follows:

Definition 18 (Strong Privacy). *An* ESSS *is strongly private, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\epsilon(\cdot)$ *such that:*

$$\mathsf{Pr}\left[\begin{array}{c}(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(1^\kappa), (\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(1^\kappa), \\ b \xleftarrow{R} \{0,1\}, \mathcal{O} \leftarrow \{\mathcal{O}^{\mathsf{Sign}}(\cdot,\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}),\cdot), \mathcal{O}^{\mathsf{Sanit}}(\cdot,\cdot,\cdot,\cdot, \\ \mathsf{sk}_{\mathsf{san}}), \mathcal{O}^{\mathsf{Proof}}(\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}),\cdot), \mathcal{O}^{\mathsf{LoRSanit}}(\cdot,\cdot,\cdot,(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), \\ (\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}), b)\}, b^* \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) \ : \ b = b^* \end{array}\right] \leq \frac{1}{2} + \epsilon(\kappa),$$

where the oracles $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{O}^{\mathsf{Sanit}}$ *and* $\mathcal{O}^{\mathsf{Proof}}$ *are defined as in Definition 10. The oracle* $\mathcal{O}^{\mathsf{LoRSanit}}$ *is defined as follows:*

$\mathcal{O}^{\mathsf{LoRSanit}}((\mathsf{m}_0, \mathsf{MOD}_0, \mathsf{V}_0), (\mathsf{m}_1, \mathsf{MOD}_1, \mathsf{V}_1), \mathsf{ADM}, (\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), (\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}), b)$:

1: *If* $\mathsf{MOD}_0 \not\preceq (\mathsf{ADM}, \mathsf{V}_0) \ \vee \ \mathsf{MOD}_1 \not\preceq (\mathsf{ADM}, \mathsf{V}_1)$*, return* $\perp$.
2: *If* $(\mathsf{m}_0, \mathsf{MOD}_0, \mathsf{ADM}, \mathsf{V}_0) \not\equiv (\mathsf{m}_1, \mathsf{MOD}_1, \mathsf{ADM}, \mathsf{V}_1)$*, return* $\perp$.
3: *Compute* $(\sigma_b, \mathsf{san}_b) \leftarrow \mathsf{Sign}(\mathsf{m}_b, \mathsf{ADM}, \mathsf{V}_b, (\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), \mathsf{pk}_{\mathsf{san}})$.
4: *Return* $(\mathsf{m}'_b, \sigma'_b) \leftarrow \mathsf{Sanit}((\mathsf{m}_b, \sigma_b), \mathsf{MOD}_b, \mathsf{san}_b, \mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}})$.

Note that for answers from the oracle $\mathcal{O}^{\mathsf{LoRSanit}}$*, the oracle* $\mathcal{O}^{\mathsf{Sanit}}$ *is restricted to queries for modifications which are covered by both set limitations* V_0 *and* V_1*, which were initially submitted to* $\mathcal{O}^{\mathsf{LoRSanit}}$.

Theorem 1. *Privacy is strictly weaker than strong privacy, while (strong) unlinkability is strictly stronger than strong privacy.*

As mentioned in [9], the extension of the model regarding `LimitSet` does not influence the relations of the properties shown in [4]. That is, *unforgeability* is implied by *accountability*, *(proof-restricted) privacy* is implied by *(proof-restricted) transparency* and *immutability* is still independent of the other properties. What remains for the proof of Theorem 1 is to unveil the relations of *strong privacy* to the other privacy related notions. We subsequently prove a number of lemmas to finally obtain the desired result.

Lemma 1. *Not every transparent* ESSS *is strongly private.*

We prove Lemma 1 by counterexample.

[7] In [22], a security notion called strong privacy has been introduced for *plain* sanitizable signatures. Our notion of strong privacy is unrelated to their notion and does not conflict with their notion as ours is only meaningful in context of ESSS.

Proof. Let us consider an instantiation of Scheme 1 with a *correct*, *unforgeable*, *immutable*, *private*, *(proof-restricted) transparent* and *accountable* sanitizable signature scheme. Further, assume that the accumulator scheme is distinguishable. Then, an adversary against the indistinguishability implies an adversary against strong privacy. □

From this proof, we can straight forwardly derive:

Corollary 1. *Not every private* ESSS *is strongly private.*

To show that strong privacy is a strictly stronger notion than privacy, we additionally need to show that the following lemma holds.

Lemma 2. *Every strongly private* ESSS *is also private.*

To prove this, we show that we can construct an efficient adversary $\mathcal{A}^{\mathsf{SP}}$ against strong privacy using an efficient adversary $\mathcal{A}^{\mathsf{P}}$ against privacy.

Proof. $\mathcal{A}^{\mathsf{SP}}$ simply forwards the calls to the oracles $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sanit}}, \mathcal{O}^{\mathsf{Proof}}$, whereas the oracle $\mathcal{O}^{\mathsf{LoRSanit}}$ is simulated as follows: Upon every query $(\mathsf{m}_0,\ \mathsf{MOD}_0)$, $(\mathsf{m}_1, \mathsf{MOD}_1)$, ADM of $\mathcal{A}^{\mathsf{P}}$, $\mathcal{A}^{\mathsf{SP}}$ internally chooses random set limitations V such that $\mathsf{MOD}_0 \preceq (\mathsf{ADM},\ \mathsf{V})$, $\mathsf{MOD}_1 \preceq (\mathsf{ADM}, \mathsf{V})$. Then $\mathcal{A}^{\mathsf{SP}}$ forwards the query $(\mathsf{m}_0, \mathsf{MOD}_0, \mathsf{V})$, $(\mathsf{m}_1, \mathsf{MOD}_1, \mathsf{V})$, ADM to its own $\mathcal{O}^{\mathsf{LoRSanit}}$ oracle and returns the result to $\mathcal{A}^{\mathsf{P}}$. Eventually, $\mathcal{A}^{\mathsf{P}}$ outputs a bit b which is forwarded by $\mathcal{A}^{\mathsf{SP}}$. It is easy to see that the winning probability of $\mathcal{A}^{\mathsf{SP}}$ is identical to that of $\mathcal{A}^{\mathsf{P}}$. □

Subsequently, we show that unlinkability is strictly stronger than strong privacy.

Lemma 3. *Not every strongly private* ESSS *is (strongly) unlinkable.*

We prove Lemma 3 by counterexample.

Proof. Let us consider an instantiation of Scheme 1 with a *correct*, *unforgeable*, *immutable*, *private*, *(proof-restricted) transparent* and *accountable* sanitizable signature scheme which does *not* fulfill unlinkability. By Theorem 3, we can extend it to be strongly private by using an indistinguishable accumulator. □

Lemma 4. *Every unlinkable* ESSS *is also strongly private.*

To prove Lemma 4, we show that we can construct an efficient adversary $\mathcal{A}^{\mathsf{U}}$ against unlinkability using an efficient adversary $\mathcal{A}^{\mathsf{SP}}$ against strong privacy.

Proof. Likewise to the proof of Lemma 2, $\mathcal{A}^{\mathsf{U}}$ simply forwards the calls to the oracles $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sanit}}, \mathcal{O}^{\mathsf{Proof}}$, whereas the oracle $\mathcal{O}^{\mathsf{LoRSanit}}$ is simulated as follows: Upon every query $(\mathsf{m}_0,\ \mathsf{MOD}_0, \mathsf{V}_0)$, $(\mathsf{m}_1, \mathsf{MOD}_1, \mathsf{V}_1)$, ADM of $\mathcal{A}^{\mathsf{SP}}$, $\mathcal{A}^{\mathsf{U}}$ obtains $(\sigma_0, \mathsf{san}_0) \leftarrow \mathcal{O}^{\mathsf{Sign}}(\mathsf{m}_0, \mathsf{ADM}, \mathsf{V}_0)$, $(\sigma_1, \mathsf{san}_1) \leftarrow \mathcal{O}^{\mathsf{Sign}}(\mathsf{m}_1, \mathsf{ADM}, \mathsf{V}_1)$ using its own $\mathcal{O}^{\mathsf{Sign}}$ oracle. Then $\mathcal{A}^{\mathsf{U}}$ forwards the query $(\mathsf{m}_0,\ \mathsf{MOD}_0, \mathsf{san}_0, \sigma_0)$, $(\mathsf{m}_1, \mathsf{MOD}_1, \mathsf{san}_1,\ \sigma_1)$, ADM to its own $\mathcal{O}^{\mathsf{LoRSanit}}$ oracle and returns the result to $\mathcal{A}^{\mathsf{SP}}$. Eventually, $\mathcal{A}^{\mathsf{SP}}$ outputs a bit b which is forwarded by $\mathcal{A}^{\mathsf{U}}$. It is easy to see that the winning probability of $\mathcal{A}^{\mathsf{U}}$ is identical to that of $\mathcal{A}^{\mathsf{SP}}$. □

Taking all the above results together, Theorem 1 follows.

5 Black-Box Extension of Sanitizable Signatures

Provably secure existing constructions of ESSS build up on concrete existing sanitizable signature schemes. As it turns out, we can even obtain a more general result, i.e., we obtain an ESSS that only makes black-box use of sanitizable signatures in the model of [4, 18] and secure accumulators. The so obtained black-box construction of an ESSS then fulfills all the security notions of the underlying sanitizable signature scheme.

Before we continue, we recall the general paradigm for instantiating `LimitSet` (cf. [9, 21]).

Paradigm 1. *For each* `LimitSet` *block, use a secure accumulator* ACC *to accumulate the set of admissible replacements. The respective message blocks are then replaced with the corresponding accumulator value, i.e., the accumulators are included in the same way as fixed message blocks. Conversely, the actually chosen message blocks for each* `LimitSet` *block are included in the same way as variable message blocks (since they change on every sanitization). Finally, the signature is augmented by the witnesses corresponding to the actual message blocks, while the remaining witnesses are only known to the signer and the sanitizer.*

We introduce our generic construction (that follows Paradigm 1) in Scheme 1, where we use ($\mathbf{KeyGen}_{\mathsf{sig}}$, $\mathbf{KeyGen}_{\mathsf{san}}$, **Sign**, **Sanit**, **Verify**, **Proof**, **Judge**) to denote the algorithms of the underlying sanitizable signature scheme. We define two operators ϕ and ψ to manipulate sets $S = \{(k_1, v_1), \ldots, (k_n, v_n)\}$ of key-value pairs. Thereby, we assume the keys $k_1, \ldots, k_n$ to be unique. The operator $\phi(\cdot, \cdot)$ takes a key k and a set S, obtains the tuple (k_i, v_i) with $k = k_i$ from S, and returns v_i. If no such tuple exists, $\perp$ is returned. Similarly, the operator $\psi(\cdot, \cdot, \cdot)$, takes a key k, a value v_i' and a set S and obtains the tuple (k_i, v_i) with $k = k_i$ from S. It returns $(S \setminus \{(k_i, v_i)\}) \cup \{(k_i, v_i')\}$ and $\perp$ if no such tuple exists.

We will prove the security of Scheme 1 using similar arguments as in [9], but relying on the abstract model of [4, 18], instead of specific properties of the used sanitizable signature scheme.

Theorem 2. *When instantiating Scheme 1 with a sanitizable signature scheme that provides security properties Σ in the model of [4, 18] and a secure accumulator scheme, one obtains an* ESSS *that provides security properties Σ.*

We prove Theorem 2 in Appendix A. Furthermore, we emphasize that—while our model includes the extensions regarding ADM from [18]—the proof does not rely on these extensions. This means that our black-box construction is also applicable to schemes in the model of [4].

Now, we discuss some observations related to the instantiation of the `LimitSet` extension using accumulators. As discussed in the previous section, it seems to be hard to design generic extensions that also preserve unlinkability [6, 8]. Furthermore, the abstract model does not consider the signer as an adversary, which gives some freedom regarding the implementations of certain algorithms and the choice of the accumulator scheme. As mentioned in Section 2.1, the

$\mathsf{KeyGen}_{\mathsf{sig}}(1^\kappa)$**:** Given a security parameter κ, run $(\mathbf{sk}_{\mathsf{sig}}, \mathbf{pk}_{\mathsf{sig}}) \leftarrow \mathbf{KeyGen}(1^\kappa)$, choose an accumulator scheme and run $(\mathsf{sk}_{\mathsf{acc}}, \mathsf{pk}_{\mathsf{acc}}) \leftarrow \mathsf{AGen}(1^\kappa)$. Finally, return $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow ((\mathbf{sk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{acc}}), (\mathbf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{acc}}))$.

$\mathsf{KeyGen}_{\mathsf{san}}(1^\kappa)$**:** Given a security parameter κ, return $(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) = (\mathbf{sk}_{\mathsf{san}}, \mathbf{pk}_{\mathsf{san}}) \leftarrow \mathbf{KeyGen}_{\mathsf{san}}(1^\kappa)$.

$\mathsf{Sign}(\mathsf{m}, \mathsf{ADM}, \mathsf{V}, (\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), \mathsf{pk}_{\mathsf{san}})$**:** Given $\mathsf{m} = (\mathsf{m}_i)_{i=1}^n$, $\mathsf{ADM} = (\mathsf{B}_i)_{i=1}^n$, $\mathsf{V} = \{(i, \mathsf{M}_i) : \mathsf{B}_i = \mathtt{lim} \wedge \mathsf{M}_i \subset \bigcup_{j=0}^{l_i} \{0,1\}^j\}$, $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}})$ and $\mathsf{pk}_{\mathsf{san}}$, this algorithm sets $\mathcal{V}, \mathcal{W} \leftarrow \emptyset$ and computes
for $i = 1 \ldots n$ **if** $\mathsf{B}_i = \mathtt{lim}$ **do**:
$\mathsf{M}_i \leftarrow \phi(i, \mathsf{V}), \mathsf{acc}_i \leftarrow \mathsf{AEval}((\mathsf{sk}_{\mathsf{acc}}, \mathsf{pk}_{\mathsf{acc}}), \mathsf{M}_i), \mathcal{W}_i \leftarrow \emptyset$,
$\forall v_j \in \mathsf{M}_i : \mathsf{wit}_{i_j} \leftarrow \mathsf{AWitCreate}((\mathsf{sk}_{\mathsf{acc}}, \mathsf{pk}_{\mathsf{acc}}), \mathsf{acc}_i, \mathsf{M}_i, v_j), \mathcal{W}_i \leftarrow \mathcal{W}_i \cup \{(v_j, \mathsf{wit}_{i_j})\}$
$\mathcal{V}_i \leftarrow (i, (\phi(\mathsf{m}_i, \mathcal{W}_i), \mathsf{acc}_i)), \mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{V}_i, \mathcal{W} \leftarrow \mathcal{W} \cup \{(i, \mathcal{W}_i)\}$,
$\mathsf{B}_i \leftarrow \mathtt{var}, \mathsf{m} \leftarrow \mathsf{m}||(\mathsf{acc}_i, i), \mathsf{ADM} \leftarrow \mathsf{ADM}||(\mathtt{fix})$.
endfor.
Next, it computes $\hat{\sigma} \leftarrow \mathbf{Sign}(\mathsf{m}, \mathsf{ADM}, (\mathbf{sk}_{\mathsf{sig}}, \mathbf{pk}_{\mathsf{sig}}), \mathbf{pk}_{\mathsf{san}})$. Finally it sets $\sigma \leftarrow (\hat{\sigma}, \mathcal{V})$ and $\mathsf{san} \leftarrow (\emptyset, \mathcal{W})$ and returns (σ, san), or $\perp$ if any of the calls to ϕ, ψ or **Sign** fails.

$\mathsf{Sanit}((\mathsf{m}, \sigma), \mathsf{MOD}, \mathsf{san}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}})$**:** Given $(\mathsf{m}, \sigma) = ((\mathsf{m}_i)_{i=1}^n, \sigma)$, $\mathsf{MOD} = \{(i, \mathsf{m}'_i)\}^t$, san, $\mathsf{pk}_{\mathsf{sig}}$ and $\mathsf{sk}_{\mathsf{san}}$, this algorithm computes
for $i = 1 \ldots n$ **if** $\mathsf{B}_i = \mathtt{var} \wedge \perp \neq \phi(i, \mathcal{W})$ **do**:
$\mathcal{W}_i \leftarrow \phi(i, \mathcal{W}), \mathsf{wit} \leftarrow \phi(\mathsf{m}'_i, \mathcal{W}_i), (\mathsf{wit}_{i_j}, \mathsf{acc}_i) \leftarrow \phi(i, \mathcal{V}), \mathcal{V}' \leftarrow \psi(i, (\mathsf{wit}, \mathsf{acc}_i), \mathcal{V})$.
endfor.
Finally, it computes $\hat{\sigma} \leftarrow \mathbf{Sanit}(\mathit{Ext}(\mathsf{m}, \sigma), \mathsf{MOD}, \mathbf{pk}_{\mathsf{sig}}, \mathbf{sk}_{\mathsf{san}})$ and returns $\sigma = (\hat{\sigma}, \mathcal{V})$, or $\perp$ if any of the calls to ϕ, ψ or **Sanit** fails.

$\mathsf{Verify}((\mathsf{m}, \sigma), \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$**:** Given $(\mathsf{m}, \sigma) = ((\mathsf{m}_i)_{i=1}^n, \sigma)$, $\mathsf{pk}_{\mathsf{sig}}$ and $\mathsf{pk}_{\mathsf{san}}$, this algorithm verifies whether $\mathbf{Verify}(\mathit{Ext}(\mathsf{m}, \sigma), \mathbf{pk}_{\mathsf{sig}}, \mathbf{pk}_{\mathsf{san}}) = \mathtt{false}$ and returns `false` if so. Otherwise, it computes
for $i = 1 \ldots n$ **if** $\mathsf{B}_i = \mathtt{var} \wedge \perp \neq \phi(i, \mathcal{V})$ **do**:
$(\mathsf{wit}_{i_j}, \mathsf{acc}_i) \leftarrow \phi(i, \mathcal{V})$, **if**$[\mathsf{AVerify}(\mathsf{pk}_{\mathsf{acc}}, \mathsf{acc}_i, \mathsf{wit}_{i_j}, \mathsf{m}_i) = \mathtt{false}]$ { **return** `false` }.
endfor.
Finally, it returns `true`.

$\mathsf{Proof}((\mathsf{m}, \sigma), \{(\mathsf{m}_j, \sigma_j)\}_{j=0}^q, (\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), \mathsf{pk}_{\mathsf{san}})$**:** Return $\mathbf{Proof}(\mathit{Ext}(\mathsf{m}, \sigma), \{\mathit{Ext}(\mathsf{m}_j, \sigma_j)\}_{j=0}^q, (\mathbf{sk}_{\mathsf{sig}}, \mathbf{pk}_{\mathsf{sig}}), \mathbf{pk}_{\mathsf{san}})$.

$\mathsf{Judge}((\mathsf{m}, \sigma), \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \pi)$**:** Return $\mathbf{Judge}(\mathit{Ext}(\mathsf{m}, \sigma), \mathbf{pk}_{\mathsf{sig}}, \mathbf{pk}_{\mathsf{san}}, \pi)$.

$\mathit{Ext}(\mathsf{m}, \sigma)$**:** On input $(\mathsf{m}, \sigma) = ((\mathsf{m}_i)_{i=1}^n, \sigma)$,
for $i = 1 \ldots n$ **do**:
$(\mathsf{wit}_{i_j}, \mathsf{acc}_i) \leftarrow \phi(i, \mathcal{V})$, **if** $[(\mathsf{wit}_{i_j}, \mathsf{acc}_i) \neq \perp]$ { set $\mathsf{m} \leftarrow \mathsf{m}||(\mathsf{acc}_i, i)$ }.
endfor.
Return (m, σ).

Scheme 1: Black-box construction of ESSS from any sanitizable signature scheme.

abstract model of accumulators assumes a trusted setup. It is, however, beneficial that the signer runs the AGen algorithm to be able to perform more efficient updates using the trapdoor. As a side effect, this also means that the signer is later able to extend the limited sets by making use of the trapdoor in the fashion of [25]. If this feature is unwanted, a TTP can run the AGen algorithm and publish $\mathsf{pk}_{\mathsf{acc}}$ as a common reference string.

5.1 Obtaining Strong Privacy via a Black-Box Construction

Now we show how strongly private ESSS can be constructed from private sanitizable signature schemes in a black-box fashion. Basically, this can be achieved by applying the conversion in Scheme 1 and instantiating `LimitSet` using an accumulator that provides the indistinguishability property.

Theorem 3. *Let* ESSS *obtained using Scheme 1 be private and* (AGen, AEval, AWitCreate, AVerify) *be an indistinguishable accumulator, then* ESSS *is strongly private.*

Proof. We prove the theorem above by using a sequence of games. Thereby, we denote the event that the adversary wins Game i by S_i.

Game 0: The original strong privacy game.
Game 1: As in the original game, but we modify the oracle $\mathcal{O}^{\mathsf{LoRSanit}}$ to firstly compute $\mathsf{V} \leftarrow \mathsf{V}_0 \cap \mathsf{V}_1$ and to set $\mathsf{V}_0 \leftarrow \mathsf{V}$, $\mathsf{V}_1 \leftarrow \mathsf{V}$.

Transition Game 0 → Game 1: A distinguisher between Game 0 and Game 1 is a distinguisher for the indistinguishability game of the accumulator.

In Game 1, the signatures are computed with respect to $\mathsf{V}_0 \cap \mathsf{V}_1$ in $\mathcal{O}^{\mathsf{LoRSanit}}$. This means that the `LimitSet` related values are independent of the bit b (similar as when randomly choosing V). Thus, from the adversary's viewpoint, Game 1 is equivalent to the conventional privacy game, meaning that $\mathsf{Pr}[S_1] \leq \frac{1}{2} + \epsilon_{\mathsf{priv}}(\kappa)$. Furthermore, we know that the distinguishing probability between Game 0 and Game 1 is equivalent to the indistinguishability advantage of the accumulators, i.e., $|\mathsf{Pr}[S_0] - \mathsf{Pr}[S_1]| = k \cdot \epsilon_{\mathsf{ind}}(\kappa)$, where k is the number of `LimitSet` blocks.[8] In further consequence, this shows that the advantage of an adversary to win the strong privacy game is negligible and bounded by $\mathsf{Pr}[S_0] \leq \frac{1}{2} + \epsilon_{\mathsf{priv}}(\kappa) + k \cdot \epsilon_{\mathsf{ind}}(\kappa)$. □

We also note that it might be an option to use cfw-indistinguishable accumulators instead of indistinguishable accumulators if the accumulation domain is large enough that the chosen random value x_r can not be efficiently guessed. This would resemble the suggestion of [21], who informally mentioned that additionally accumulating a random value might prevent the adversary from guessing the set limitations.

6 Conclusion

In this paper we propose the notion of strong privacy for ESSS, which, in contrast to the privacy notion for ESSS of [9] covers privacy for the `LimitSet` extension in the original sense of sanitizable signatures. From a practical perspective, our black-box constructions nicely combine with existing schemes in the model of [4, 18]. Thus, existing implementations of schemes in these models directly yield a basis to instantiate our proposed extensions with relatively low effort, while preserving the efficiency of the underlying schemes. Conversely, it is still an open issue to construct efficient (strongly) unlinkable ESSS or to come up with a generic extension to construct such schemes from existing unlinkable sanitizable signature schemes.

[8] For compactness, we exchange all accumulators in a single game change and note that it is straight forward to unroll the exchange of the accumulators to k simple game changes.

References

1. Anandan, B., Clifton, C., Jiang, W., Murugesan, M., Pastrana-Camacho, P., Si, L.: t-Plausibility: Generalizing words to desensitize text. Transactions on Data Privacy (3) (2012)
2. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable signatures. In: ESORICS'05. LNCS (2005)
3. Bauer, D., Blough, D.M., Mohan, A.: Redactable signatures on data with dependencies and their application to personal health records. In: ACM WPES 2009 (2009)
4. Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of sanitizable signatures revisited. In: PKC'09. LNCS (2009)
5. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Santizable signatures: How to partially delegate control for authenticated data. In: BIOSIG 2009 (2009)
6. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Unlinkability of sanitizable signatures. In: PKC 2010 (2010)
7. Brzuska, C., Pöhls, H.C., Samelin, K.: Non-interactive public accountability for sanitizable signatures. In: EuroPKI 2012 (2012)
8. Brzuska, C., Pöhls, H.C., Samelin, K.: Efficient and perfectly unlinkable sanitizable signatures without group signatures. In: EuroPKI 2013 (2013)
9. Canard, S., Jambert, A.: On extended sanitizable signature schemes. In: CT-RSA'10. LNCS (2010)
10. Canard, S., Jambert, A., Lescuyer, R.: Sanitizable signatures with several signers and sanitizers. In: AFRICACRYPT 2012 (2012)
11. Canard, S., Laguillaumie, F., Milhau, M.: Trapdoor sanitizable signatures and their application to content protection. In: ACNS 2008 (2008)
12. Canard, S., Lescuyer, R.: Protecting privacy by sanitizing personal data: a new approach to anonymous credentials. In: ASIA CCS '13 (2013)
13. Chakaravarthy, V.T., Gupta, H., Roy, P., Mohania, M.K.: Efficient techniques for document sanitization. In: ACM CIKM 2008 (2008)
14. Chang, E., Xu, J.: Remote integrity check with dishonest storage server. In: ESORICS 2008 (2008)
15. Chow, R., Oberst, I., Staddon, J.: Sanitization's slippery slope: the design and study of a text revision assistant. In: SOUPS 2009. ACM (2009)
16. Derler, D., Hanser, C., Slamanig, D.: Revisiting cryptographic accumulators, additional properties and relations to other primitives. In: CT-RSA'15. LNCS (2015)
17. Fleischhacker, N., Krupp, J., Malavolta, G., Schneider, J., Schröder, D., Simkin, M.: Efficient unlinkable sanitizable signatures from signatures with rerandomizable keys. Cryptology ePrint Archive, Report 2015/395 (2015)
18. Gong, J., Qian, H., Zhou, Y.: Fully-secure and practical sanitizable signatures. In: InsCrypt'10. LNCS (2011)
19. Haber, S., Hatano, Y., Honda, Y., Horne, W.G., Miyazaki, K., Sander, T., Tezoku, S., Yao, D.: Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In: ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008 (2008)
20. Johnson, R., Molnar, D., Song, D.X., Wagner, D.: Homomorphic signature schemes. In: CT-RSA (2002)
21. Klonowski, M., Lauks, A.: Extended sanitizable signatures. In: ICISC'06. LNCS (2006)

22. de Meer, H., Pöhls, H.C., Posegga, J., Samelin, K.: On the relation between redactable and sanitizable signature schemes. In: ESSoS 2014 (2014)
23. Miyazaki, K., Iwamura, M., Matsumoto, T., Sasaki, R., Yoshiura, H., Tezuka, S., Imai, H.: Digitally signed document sanitizing scheme with disclosure condition control. IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences (1) (2005)
24. Pöhls, H.C., Karwe, M.: Redactable signatures to control the maximum noise for differential privacy in the smart grid. In: SmartGridSec 2014 (2014)
25. Pöhls, H.C., Samelin, K.: On updatable redactable signatures. In: ACNS 2014 (2014)
26. Pöhls, H.C., Samelin, K., Posegga, J.: Sanitizable signatures in XML signature - performance, mixing properties, and revisiting the property of transparency. In: ACNS 2011 (2011)
27. Slamanig, D., Rass, S.: Generalizations and extensions of redactable signatures with applications to electronic healthcare. In: CMS 2010 (2010)
28. Slamanig, D., Stranacher, K., Zwattendorfer, B.: User-centric identity as a service-architecture for eids with selective attribute disclosure. In: ACM SACMAT'14 (2014)
29. Steinfeld, R., Bull, L., Zheng, Y.: Content extraction signatures. In: ICISC (2001)
30. Stranacher, K., Krnjic, V., Zefferer, T.: Trust and reliability for public sector data. In: ICBG (2013)
31. Sweeney, L.: Achieving k-anonymity privacy protection using generalization and suppression. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems (5) (2002)
32. Yum, D.H., Seo, J.W., Lee, P.J.: Trapdoor sanitizable signatures made easy. In: ACNS 2010 (2010)
33. Zwattendorfer, B., Slamanig, D.: On privacy-preserving ways to porting the austrian eid system to the public cloud. In: IFIP SEC 2013 (2013)

A Proof of Theorem 2

Proof. Subsequently, we prove Theorem 2:

Correctness: Straightforwardly follows from inspection and the correctness of the underlying primitives.

Unforgeability: Assume an efficient adversary $\mathcal{A}$ against unforgeability. We show how this adversary can be turned into an efficient adversary $\mathcal{B}$ against unforgeability of the underlying sanitizable signature scheme SSS. To do so, we describe a reduction $\mathcal{R}$ such that $(\mathcal{A}, \mathcal{R})$ form $\mathcal{B}$. Firstly, $\mathcal{R}$ obtains $\mathbf{pk}_{\mathsf{sig}}$ and $\mathbf{pk}_{\mathsf{san}}$ from the challenger, runs $(\mathsf{sk}_{\mathsf{acc}}, \mathsf{pk}_{\mathsf{acc}}) \leftarrow \mathsf{AGen}(1^\kappa)$ and starts $\mathcal{A}$ on $((\mathbf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{acc}}), \mathsf{pk}_{\mathsf{san}})$. $\mathcal{R}$ implements the oracles by computing the accumulator related parts itself and for the rest it uses the oracles of the unforgeability challenger of the SSS. Now, by definition, $\mathcal{A}$ outputs $(\mathsf{m}^*, \sigma^*) = (\mathsf{m}^*, (\hat{\sigma}, \mathcal{V}))$ such that $\mathsf{Verify}(\mathsf{m}^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) = \mathtt{true} \wedge (\mathsf{m}^*, \mathsf{ADM}^*, \mathsf{V}^*, \mathsf{pk}_{\mathsf{san}}) \notin L^{\mathsf{Sign}} \wedge ((\mathsf{m}^*, \sigma^*), \mathsf{ADM}^*, \mathsf{pk}_{\mathsf{sig}}) \notin L^{\mathsf{Sanit}}$. This means that $\mathcal{R}$ can output $(Ext(\mathsf{m}^*, \sigma^*), \hat{\sigma})$ as a forgery for SSS. □

Remark 1. Observe that the chosen message block per `LimitSet` block is also included as variable element, while the accumulators acc_i together with the positions i of the `LimitSet` blocks in the message are treated as additional fixed elements (cf. *Ext* in Scheme 1). Consequently, *every* forgery is a forgery for the underlying SSS scheme.

Immutability: Assume an efficient adversary $\mathcal{A}$ against immutability. We show how this adversary can be turned (1) into an efficient adversary $\mathcal{B}_1$ against immutability of the underlying sanitizable signature SSS scheme, or (2) into an efficient adversary $\mathcal{B}_2$ against the collision freeness of the underlying accumulator. To do so, we describe two reductions $\mathcal{R}_1, \mathcal{R}_2$ such that $\mathcal{B}_i = (\mathcal{A}, \mathcal{R}_i), i \in \{1, 2\}$.

(1) $\mathcal{R}_1$ obtains $\mathbf{pk}_{\mathsf{sig}}$ from the challenger, runs $(\mathsf{sk}_{\mathsf{acc}}, \mathsf{pk}_{\mathsf{acc}}) \leftarrow \mathsf{AGen}(1^\kappa)$ and starts $\mathcal{A}$ on $(\mathbf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{acc}})$. $\mathcal{R}_1$ implements the oracles by computing the accumulator related parts itself and for the rest it uses the oracles of the immutability challenger of the SSS. Eventually, $\mathcal{A}$ outputs $(\mathsf{pk}^*_{\mathsf{san}}, \mathsf{m}^*, \sigma^*) = (\mathbf{pk}^*_{\mathsf{san}}, \mathsf{m}^*, (\hat{\sigma}, \mathcal{V}))$ such that $\mathsf{Verify}(\mathsf{m}^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}) = \mathtt{true} \wedge ((\cdot, \cdot, \cdot, \mathsf{pk}^*_{\mathsf{san}}) \notin L^{\mathsf{Sign}} \vee \nexists\, \mathsf{m}^{* \,(\mathsf{ADM}^*, \mathsf{V}^*)}\!\!\preceq\, \mathsf{m} : (\mathsf{m}, \mathsf{ADM}^*, \mathsf{V}^*, \mathsf{pk}^*_{\mathsf{san}}) \in L^{\mathsf{Sign}})$. If $\nexists\, \mathsf{m}^{* \,(\mathsf{ADM}^*, \mathsf{V}^*)}\!\!\preceq\, \mathsf{m} : (\mathsf{m}, \mathsf{ADM}^*, \mathsf{V}^*, \mathsf{pk}^*_{\mathsf{san}}) \in L^{\mathsf{Sign}})$ because of a modification of a `LimitSet` element that is not covered by V^*, then $\mathcal{R}_1$ aborts (since we are in the other case). Otherwise, $\mathcal{R}_1$ can output $(\mathbf{pk}^*_{\mathsf{san}}, \mathit{Ext}(\mathsf{m}^*, \sigma^*), \hat{\sigma})$ and wins the immutability game of SSS. □

(2) $\mathcal{R}_2$ obtains $\mathsf{pk}_{\mathsf{acc}}$ from the challenger, runs $(\mathbf{sk}_{\mathsf{sig}}, \mathbf{pk}_{\mathsf{sig}}) \leftarrow \mathbf{KeyGen}(1^\kappa)$ and starts $\mathcal{A}$ on $(\mathbf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{acc}})$. $\mathcal{R}_2$ can simulate the oracles by computing the SSS related parts itself and calling the respective oracles for the accumulator related computations. $\mathcal{R}_2$ keeps track of the accumulators, contained sets and (optionally) the used randomizers obtained from the oracles via a list L_{acc}. Eventually, $\mathcal{A}$ outputs $(\mathsf{pk}^*_{\mathsf{san}}, \mathsf{m}^*, \sigma^*) = (\mathbf{pk}^*_{\mathsf{san}}, \mathsf{m}^*, (\hat{\sigma}, \mathcal{V}))$ such that $\mathsf{Verify}(\mathsf{m}^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}) = \mathtt{true} \wedge ((\cdot, \cdot, \cdot, \mathsf{pk}^*_{\mathsf{san}}) \notin L^{\mathsf{Sign}} \vee \nexists\, \mathsf{m}^{* \,(\mathsf{ADM}^*, \mathsf{V}^*)}\!\!\preceq\, \mathsf{m} : (\mathsf{m}, \mathsf{ADM}^*, \mathsf{V}^*, \mathsf{pk}^*_{\mathsf{san}}) \in L^{\mathsf{Sign}})$. If $(\cdot, \cdot, \cdot, \mathsf{pk}^*_{\mathsf{san}}) \notin L^{\mathsf{Sign}}$, $\mathcal{R}_2$ aborts. If $\nexists\, \mathsf{m}^{* \,(\mathsf{ADM}^*, \mathsf{V}^*)}\!\!\preceq\, \mathsf{m} : (\mathsf{m}, \mathsf{ADM}^*, \mathsf{V}^*, \mathsf{pk}^*_{\mathsf{san}}) \in L^{\mathsf{Sign}})$ because of a modification of a `LimitSet` element that is not covered by V^*, we know that there is at least one tuple $(i, (\mathsf{wit}_{i_j}, \mathsf{acc}_i)) \in \mathcal{V}$ such that $\mathsf{AVerify}(\mathsf{pk}_{\mathsf{acc}}, \mathsf{acc}_i, \mathsf{wit}_{i_j}, \mathsf{m}_i) = \mathtt{true}$ but $\mathsf{m}_i \notin \mathsf{M}_i$, where M_i is the set contained in acc_i. Now, $\mathcal{R}_2$ can look up the set M_i and the corresponding randomizer r_i in L_{acc} and return $(\mathsf{wit}_{i_j}, \mathsf{m}_i, \mathsf{M}_i, r_i)$ as a collision for the accumulator. Otherwise, $\mathcal{R}_2$ aborts (since we are in the other case). □

Privacy: Assume an efficient adversary $\mathcal{A}$ against privacy. We show how this adversary can be turned into an efficient adversary $\mathcal{B}$ against privacy of the underlying sanitizable signature scheme SSS. To do so, we describe a reduction $\mathcal{R}$ such that $(\mathcal{A}, \mathcal{R})$ form $\mathcal{B}$. Firstly, $\mathcal{R}$ obtains $\mathbf{pk}_{\mathsf{sig}}$ and $\mathbf{pk}_{\mathsf{san}}$ from the challenger, runs $(\mathsf{sk}_{\mathsf{acc}}, \mathsf{pk}_{\mathsf{acc}}) \leftarrow \mathsf{AGen}(1^\kappa)$ and starts $\mathcal{A}$ on $((\mathbf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{acc}}), \mathsf{pk}_{\mathsf{san}})$. $\mathcal{R}$ implements the oracles by computing the accumulator related parts itself and for the rest it uses the oracles of the privacy challenger of the SSS. Eventually $\mathcal{A}$ outputs a bit b^*, which can be used by $\mathcal{R}$ to win the privacy game of

the SSS, where the winning probability is that of $\mathcal{A}$. Now, we argue why this is the case: In the privacy game, the set limitations V are internally chosen in $\mathcal{O}^{\mathsf{LoRSanit}}$ such that they are compatible with both submitted challenge messages. This, means that the `LimitSet` related signature components are independent of the actual bit b.

Transparency: Assume an efficient adversary $\mathcal{A}$ against transparency. We show how this adversary can be turned into an efficient adversary $\mathcal{B}$ against transparency of the underlying sanitizable signature scheme SSS. To do so, we describe a reduction $\mathcal{R}$ such that $(\mathcal{A}, \mathcal{R})$ form $\mathcal{B}$. Firstly, $\mathcal{R}$ obtains $\mathbf{pk}_{\mathsf{sig}}$ and $\mathbf{pk}_{\mathsf{san}}$ from the challenger, runs $(\mathsf{sk}_{\mathsf{acc}}, \mathsf{pk}_{\mathsf{acc}}) \leftarrow \mathsf{AGen}(1^{\kappa})$ and starts $\mathcal{A}$ on $((\mathbf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{acc}}), \mathsf{pk}_{\mathsf{san}})$. $\mathcal{R}$ implements the oracles by computing the accumulator related parts itself and for the rest uses the oracles of the transparency challenger of the SSS. Eventually, $\mathcal{A}$ outputs a bit b^* and $\mathcal{R}$ forwards this bit to the transparency challenger of the SSS and wins the game—the winning probability is that of $\mathcal{A}$. To see this, observe that in $\mathcal{O}^{\mathsf{Sign/Sanit}}$ the same set limitations V are used in both, the signed and the sanitized message.

Accountability: Subsequently, we prove sanitizer- and signer-accountability:

Sanitizer-Accountability: Assume an efficient adversary $\mathcal{A}$ against sanitizer-accountability. We show how this adversary can be turned into an efficient adversary $\mathcal{B}$ against sanitizer-accountability of the underlying sanitizable signature scheme SSS. To do so, we describe a reduction $\mathcal{R}$ such that $(\mathcal{A}, \mathcal{R})$ form $\mathcal{B}$. Firstly, $\mathcal{R}$ obtains $\mathbf{pk}_{\mathsf{sig}}$ from the challenger, runs $(\mathsf{sk}_{\mathsf{acc}}, \mathsf{pk}_{\mathsf{acc}}) \leftarrow \mathsf{AGen}(1^{\kappa})$ and starts $\mathcal{A}$ on $(\mathbf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{acc}})$. $\mathcal{R}$ implements the oracles by computing the accumulator related parts itself and for the rest uses the oracles of the sanitizer-accountability challenger of the SSS. Now, by definition, $\mathcal{A}$ outputs $(\mathsf{pk}^*_{\mathsf{san}}, \mathsf{m}^*, \sigma^*) = (\mathbf{pk}^*_{\mathsf{san}}, \mathsf{m}^*, (\hat{\sigma}, \mathcal{V}))$ such that $\mathsf{Verify}(\mathsf{m}^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}) = \mathtt{true} \ \wedge \ (\mathsf{m}^*, \mathsf{ADM}^*, \mathsf{V}^*, \mathsf{pk}^*_{\mathsf{san}}) \notin L^{\mathsf{Sign}} \ \wedge \ \mathsf{Judge}((\mathsf{m}^*, \sigma^*), \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}, \mathsf{Proof}((\mathsf{m}^*, \sigma^*), \mathtt{SIG}, (\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}), \mathsf{pk}^*_{\mathsf{san}})) = \mathsf{sig}$. Consequently, $\mathcal{R}$ can output $(\mathbf{pk}^*_{\mathsf{san}}, Ext(\mathsf{m}^*, \sigma^*), \hat{\sigma})$ and wins the sanitizer-accountability game of the SSS (the argumentation why is analogous to the one in Remark 1).

Signer-Accountability: Assume an efficient adversary $\mathcal{A}$ against signer-accountability. We show how this adversary can be turned into an efficient adversary $\mathcal{B}$ against signer-accountability of the underlying sanitizable signature scheme SSS. To do so, we describe a reduction $\mathcal{R}$ such that $(\mathcal{A}, \mathcal{R})$ form $\mathcal{B}$. Firstly, $\mathcal{R}$ obtains $\mathbf{pk}_{\mathsf{san}}$ from the challenger and starts $\mathcal{A}$ on $\mathbf{pk}_{\mathsf{san}}$. $\mathcal{R}$ implements the oracles by computing the accumulator related parts itself and for the rest uses the oracles of the signer-accountability challenger of the SSS. Eventually, $\mathcal{A}$ outputs $(\mathsf{pk}^*_{\mathsf{sig}}, \mathsf{m}^*, \sigma^*, \pi^*) = ((\mathbf{pk}^*_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{acc}}), \mathsf{m}^*, (\hat{\sigma}, \mathcal{V}), \pi^*)$ such that $\mathsf{Verify}(\mathsf{m}^*, \sigma^*, \mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) = \mathtt{true} \ \wedge \ ((\mathsf{m}^*, \sigma^*), \mathsf{ADM}^*, \mathsf{pk}^*_{\mathsf{sig}}) \notin L^{\mathsf{Sanit}} \ \wedge \ \mathsf{Judge}((\mathsf{m}^*, \sigma^*), \mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \pi) = \mathsf{san}$. Consequently, $\mathcal{A}$ outputs $(\mathbf{pk}^*_{\mathsf{sig}}, Ext(\mathsf{m}^*, \sigma^*), \hat{\sigma}, \pi)$ and wins the signer-accountability of the SSS (the argumentation why is analogous to the one in Remark 1).

This completes the proof. □