This is an extended, revised and clarified version of the paper which appears in DPM 2015,
Data Privacy Management, and Security Assurance - 10th International Workshop, DPM 2015, Europe, Vienna, Austria, September 21-22, 2015,
Proceedings. LNCS 9481 pages 100–117.
The final authenticated version is available online at https://doi.org/10.1007/978-3-319-29883-2_7

Stronger Security for Sanitizable Signatures*

Stephan Krenn[1], Kai Samelin[2], and Dieter Sommer[3]

[1] AIT Austrian Institute of Technology GmbH, Vienna, Austria
stephan.krenn@ait.ac.at
[2] TÜV Rheinland i-sec GmbH, Hallbergmoos, Germany
kaispapers@gmail.com
[3] Ergon Informatik AG, Zurich, Switzerland
dieter.sommer@ergon.ch

Abstract. Sanitizable signature schemes ($\mathcal{SSS}$) enable a designated party (called the *sanitizer*) to alter admissible blocks of a signed message. This primitive can be used to remove or alter sensitive data from already signed messages without involvement of the original signer.
Current state-of-the-art security definitions of $\mathcal{SSS}$s only define a "weak" form of security. Namely, the unforgeability, accountability and transparency definitions are not strong enough to be meaningful in certain use-cases. We identify some of these use-cases, close this gap by introducing stronger definitions, and show how to alter an existing construction to meet our desired security level. Moreover, we clarify a small yet important detail in the state-of-the-art privacy definition. Our work allows to deploy this primitive in more and different scenarios.

1 Introduction

Traditional digital signature schemes such as RSA-PSS require that a signature σ on a message m becomes invalid as soon as a single bit of m is altered [8,25]. Contrary, many use-cases require subsequent changes to the signed data by a semi-trusted third party. As a simple example, consider a driver's license which is signed by the issuing state. To prove majority, the holder wants to remove all information but the date of birth and its picture to preserve his privacy. Obviously, having the data re-signed by the state every time the holder needs to prove its age induces too much overhead to be practical in this scenario. This constellation is widely known as the "digital document sanitization problem" [34].

Sanitizable signature schemes ($\mathcal{SSS}$) [3] address the aforementioned shortcomings. They allow for altering all signer-chosen admissible blocks $m[i]$ of a given message $m = (m[1], \ldots, m[i], \ldots, m[\ell])$ to different bitstrings $m[i]' \in \{0,1\}^*$ by the sanitizer, which holds its own private key. In particular, a sanitization of a message m creates an altered message $m' = (m[1]', \ldots, m[i]', \ldots, m[\ell]')$, where $m[i] = m[i]'$ for every non-admissible block, and a signature σ', verifying under the given public keys.

Application scenarios include secure routing, privacy-preserving handling of patient data, official document disclosure, and blank signatures [3,11,12,13,21,28].

Organization. This paper is structured as follows. The remainder of this section is devoted for pointing out the problems in current definitions, our contribution and existing related work. The required preliminaries are given in §2. The security model of $\mathcal{SSS}$s is revised in §3, while §4 contains the altered construction. We conclude our work in §5.

Our Contribution. We review the existing state-of-the-art security model for $\mathcal{SSS}$s and show that it is not sufficient for certain use-cases. We then revise the model, resulting in strictly stronger security definitions. Finally, we sketch how to alter the construction of *Bruszka* et al. [11], such that it satisfies our new definitions.

More precisely, we introduce the notions of strong transparency, strong sanitizer- and signer-accountability, strong non-interactive public accountability, and strong unforgeability. Moreover, we show that the original privacy definition may not be clear enough and we provide a clearer definition. To get a first understanding, we briefly describe each security property. The corresponding formal definitions are given in §3. We want stress that we work in the single signer/single sanitizer model. Extensions to multiple signer/multiple sanitizer environments are straightforward.

* This work was supported by the Horizon 2020 project PRISMACLOUD under grant agreement no. 644962, and the FP7 projects FutureID and AU2EU under grant agreement nos. 318424 and 611659. Parts of this work were done while the authors were at IBM Research – Zurich and the second author also at TU Darmstadt.

(Strong) Unforgeability. In $\mathcal{SSS}$s, there are two key pairs, one for the signer and one for the sanitizer. Unforgeability requires that an adversary, not having access to any of these private keys, cannot generate a signature on messages not endorsed by the signer or the sanitizer.

Our new stronger definition also requires that the adversary cannot generate *any* new signatures on its own, even with fully adaptive oracle access.

Immutability. As already mentioned before, the signer is able to define non-admissible blocks, i.e., blocks which can neither be modified by an outsider nor by the sanitizer. Immutability thus requires that even a sanitizer holding its own private key cannot change non-admissible blocks.

Privacy. A very important property is privacy. It defines that once a message has been sanitized, an outsider cannot derive any information about the original content. Clearly, privacy is necessary to have a meaningful primitive. We clarify a small detail in the current state-of-the-art definition, which may be easily overlooked. Jumping ahead, in the security game for privacy, there are algorithms used whose behavior we clarify, i.e., make explicit.

(Strong) Transparency. The property of transparency is a stronger privacy definition. In particular, if an $\mathcal{SSS}$ is transparent, an outsider not holding any private keys cannot decide whether a signature was generated by the signer or the sanitizer, if the corresponding message was never de-anonymized. In other words, transparency is the anonymity of the accountable party. Note, the naming is due to historic reasons [3]. Our stronger definition guarantees transparency, as long as no message/signature *pair* was de-anonymized. We want to stress that *Ateniese* et al. already defined strong transparency in [3]. However, *Pöhls* et al. showed that this property is not defined correctly [35]. Hence, our definition does not collide with their definition.

Unlinkability. Unlinkability is also a very strong privacy notion. It prohibits an adversary from deciding from which signature a derived signature was created.

(Strong) Signer-Accountability. As $\mathcal{SSS}$s allow to alter signed data, it must be derivable which party is responsible for a given message/signature pair. Signer-accountability thus requires that the signer cannot deny that the message originated from itself, if it was signed by the signer.

Our stronger definition also accounts for the signature, i.e., even the signer cannot blame the sanitizer for a *signature* which the sanitizer did not generate.

(Strong) Sanitizer-Accountability. Sanitizer-accountability is the counterpart to signer-accountability. In particular, the roles are reversed. This is also true for our stronger definition.

(Strong) Non-Interactive Public-Accountability. In the original definition of $\mathcal{SSS}$s, the accountable party could only be derived if the signer provides a proof π which can be generated if the signer's private key is known [9]. This contradicts legal and application requirements [12,13,29].

Non-interactive public accountability solves this problem by requiring that the accountable party can be derived without requiring any information from either the signer or the sanitizer. This property is mutually exclusive with transparency. As before, our strong definition also takes the signature into account.

Discussion. One may argue that our stronger definitions only make sense in the context of re-randomizable signatures or similar primitives. However, the (contrived) construction given in §3 shows that this is not true: there are cases which are trivially insecure in our setting, regardless of the signature scheme used. Moreover, we want to explicitly stress that, on first sight, our changes to the existing definitions seem to be rather technical and only addressing details. However, the provided examples prove that our new definitions may open a wider deployment of this primitive. In other words, this paper wants to raise awareness that certain constellations where $\mathcal{SSS}$s are used need extra precautions and guidance on how to avoid these pitfalls.

Motivation. Standard digital signatures normally have the property of existential unforgeability against chosen-message attacks (eUNF-CMA) [25]. Roughly, this definition says that an adversary cannot find a signature σ^* on a new message m^* even it has access to a signing oracle which it can query adaptively on messages of its own choice. Later, a stronger notion named "strong existential unforgeability against chosen-message attacks" (seUNF-CMA) has been

introduced [2]. In this definition, an adversary must not be able to find any message/signature pair (m^*, σ^*) which it has not seen before, i.e., the adversary must not be able to generate *any* signature σ^* on its own, even if it already knows arbitrarily many signatures on messages of its own choice, potentially including m^*. This allows for a broader use of digital signatures [2,30]. We follow the same line of research for $\mathcal{SSS}$s. In a nutshell, we now also consider the signatures in the security definitions; as $\mathcal{SSS}$s have three parties, this is an even more subtle task than for standard signatures.

Let us further elaborate: first, for unforgeability, the very same problems as for standard signatures apply. Namely, once an adversary learns a signature σ on some message m, it can potentially generate many new signatures σ_i^* on the same message m, which then would trace back to the sanitizer or signer, respectively, without this party having the chance of denying the authenticity of the message/signature pair.

Next, transparency, i.e., the anonymity of the accountable party, is only guaranteed for message/signature pairs (m_i, σ_i), where the *message* m_i has never been queried to the proof oracle (which informally allows for determining the accountable party; cf. §2). However, this means that an outsider might be able to determine the accountable party for *all* signatures on messages queried to the proof oracle, which is a problem if certain messages are signed or sanitized more than once. We clarify this statement by providing a construction secure in the old model which is "obviously" insecure in real-world scenarios in §3.

For all three accountability definitions, similar problems apply. Consider the following examples which clarify our claims:

Examples. Assume that a medical doctor signs a patient record with an $\mathcal{SSS}$. The signed medical record can be sanitized by a server. It removes identifying information from the record before giving it to the accountant, which, in turn, charges the insurance company. This procedure allows to protect the patients' privacy. The current unforgeability definition now guarantees that no party can generate signatures for *new messages*. However, the definition does not guarantee that an outsider cannot generate new signatures on *already seen* signed messages. This means that the accountant may be able to generate new signatures. If the invoice only consists of the amount due and the treatments, the accountant can thus charge more than once if no extra precautions are deployed, as, e.g., an invoice for a treatment of a broken bone is a common scenario.

As we have two "signers" and a verifier, there also exist intermediate stages, namely for all three accountability definitions. In particular, a corrupt server in our hospital example may alter messages in a way that the medical doctor is accountable, while it is not, e.g., by altering cheap treatments to expensive treatments which have existed before. Thus, the insurance company is charged more, while the accountable party for that particular *signature* may not be the party who has generated the signature. The same is true for the medical doctor: if the signer signs fresh messages which have already been output by the sanitizer, the current definition does not guarantee that the signer cannot blame the sanitizer.

For transparency it is even more problematic as for accountability and unforgeability. Again, the current definition only guarantees that transparency only holds for message/signature pairs (m_i, σ_i) for which the *message* m_i has never been "opened", i.e., the signer generated a proof which allows for tracing the accountable party. Depending on the use-case, this may also not always be desired. To be more precise, consider criminal records. In some countries, these records are deleted after ten years. If these records are signed by a national authority using an $\mathcal{SSS}$, a local municipal office can later sanitize the corresponding entries. However, once it comes to dispute over a single record and it is checked whether it had been sanitized or not, potentially all equivalent records can be traced to either the sanitizer or the signer. If, in this slightly simplified scenario, a record only contains the name and the criminal record, the privacy of all citizens with the same name would be at risk. Obviously, this contradicts the very intention of $\mathcal{SSS}$s.

We want to explicitly note that in the scenarios sketched above standard mechanisms such as also signing a unique id or the current timestamp does not help at all: if the $\mathcal{SSS}$ is unlinkable [11,13,24], such a "tag" destroys the unlinkability of the signature. Clearly, it depends on the use-case which security notions are required.

Related Work. $\mathcal{SSS}$s have originally been introduced by *Ateniese* et al. [3]. *Brzuska* et al. formalized most of the current security properties in [9]. These have been later extended for unlinkability [11,13,24] and non-interactive public accountability [12,13]. See §3 for the definitions. Some properties discussed in [9] have then been refined in [26]; namely they also consider the admissible blocks in the security games. Recently, several extensions such as limiting the sanitizer to signer-chosen values [16,22,31,35], trapdoor $\mathcal{SSS}$s (which additionally allow to add new sanitizers after signature generation by the signer) [18,36], multi-sanitizer and -signer environments [10,13,17], and sanitization of signed and encrypted data [23] have been considered. Currently, the only work considering $\mathcal{SSS}$s and data-structures more complex

than lists is [35]. Our results directly carry over to the aforementioned extended settings with only minor adjustments. Real implementations and the corresponding performance measurements of $\mathcal{SSS}$s have also been presented [12,13,19].

There exists much additional related work, e.g., proxy signatures [33]. *Ahn* et al. [1] and *Demirel* et al. [20] provide a comprehensive overview.

2 Preliminaries

Here, we present some basic notation and the general framework for $\mathcal{SSS}$s.

Notation. By $\lambda \in \mathbb{N}$ we denote the main security parameter. All algorithms implicitly take 1^λ as their first input. We write $a \leftarrow A(x)$ if a is assigned the output of algorithm A with input x. For a message $m = (m[1], \ldots, m[\ell])$, where $m[i] \in \{0,1\}^*$, we call $m[i]$ a block, while $\ell \in \mathbb{N}$ denotes the number of blocks in a message m. We call an algorithm efficient if it runs in probabilistic polynomial time in λ. All algorithms may return an exception $\perp \notin \{0,1\}^*$.

A function $\nu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is *negligible*, if it vanishes faster than every inverse polynomial. That is, for every $k \in \mathbb{N}$ there exists an $n_0 \in \mathbb{N}$ such that $\nu(n) \leq n^{-k}$ for all $n > n_0$.

Additional preliminaries are presented in Appendix A.

2.1 Sanitizable Signature Schemes

Here, we introduce the algorithms required for $\mathcal{SSS}$s and the corresponding state-of-the-art security model. The definitions are compiled from [9,12,13]. The security model is discussed and altered in §3.

Definition 1 (Sanitizable Signatures). *A sanitizable signature scheme $\mathcal{SSS}$ consists of seven efficient algorithms $(\mathsf{KGen}_{sig}, \mathsf{KGen}_{san}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ such that:*

1. *Key Generation: There is one key generation algorithm for the signer and one for the sanitizer. Both create a key pair; a private key and the corresponding public key, w.r.t. the security parameter λ:*
$$(\mathrm{pk}_{sig}, \mathrm{sk}_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^\lambda), (\mathrm{pk}_{san}, \mathrm{sk}_{san}) \leftarrow \mathsf{KGen}_{san}(1^\lambda)$$
2. *Signing: The Sign algorithm takes as input a message m, sk_{sig}, pk_{san}, as well as a description ADM of the admissibly modifiable blocks. ADM contains the set of indices of the modifiable blocks, as well as the number ℓ of blocks in m, while we assume that ADM can be uniquely and unambiguously derived from any valid message/signature pair. We write $\mathrm{ADM}(m) = 1$, if ADM is valid w.r.t. m, i.e., ADM contains the correct ℓ and all indices are in m. If $\mathrm{ADM}(m) = 0$, this algorithm returns $\perp$. It outputs a signature σ:*
$$\sigma \leftarrow \mathsf{Sign}(m, \mathrm{sk}_{sig}, \mathrm{pk}_{san}, \mathrm{ADM})$$
3. *Sanitizing: Algorithm Sanit takes a message m, modification instruction MOD, a valid signature σ, pk_{sig}, and sk_{san}. It modifies the message m according to the modification instruction MOD, which is a set containing pairs $(i, m[i]')$ for those blocks that shall be modified, meaning that $m[i]$ is replaced with $m[i]'$. We write $\mathrm{ADM}(\mathrm{MOD}) = 1$, if MOD is valid w.r.t. ADM, meaning that the indices to be modified are contained in ADM. Sanit calculates a new signature σ' for the modified message $m' \leftarrow \mathrm{MOD}(m)$. Then, Sanit outputs m' and σ':*
$$(m', \sigma') \leftarrow \mathsf{Sanit}(m, \mathrm{MOD}, \sigma, \mathrm{pk}_{sig}, \mathrm{sk}_{san})$$
4. *Verification: The Verify algorithm outputs a decision $d \in \{\mathtt{true}, \mathtt{false}\}$ verifying the correctness of a signature σ for a message m w.r.t. the public keys pk_{sig} and pk_{san}:*
$$d \leftarrow \mathsf{Verify}(m, \sigma, \mathrm{pk}_{sig}, \mathrm{pk}_{san})$$
5. *Proof: The Proof algorithm takes as input sk_{sig}, a message $m = (m[1], m[2], , \ldots, m[\ell])$, $m[i] \in \{0,1\}^*$ and a valid signature σ and a set of (polynomially many) additional message/signature pairs $\{(m_i, \sigma_i)\}$ and pk_{san}. It outputs a string $\pi \in \{0,1\}^*$ which can be used by the next algorithm (Judge) to derive the accountable party of a given message/signature pair:*
$$\pi \leftarrow \mathsf{Proof}(\mathrm{sk}_{sig}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \mathrm{pk}_{san})$$
6. *Judge: Algorithm Judge takes as input a message m, a valid signature σ, both public keys and a proof π. Note, this means that once a proof π is generated, no additional action is required with the signer or the sanitizer. It outputs a decision $d \in \{\mathtt{Sig}, \mathtt{San}\}$ indicating whether the message/signature pair has been created by the signer or a sanitizer:*
$$d \leftarrow \mathsf{Judge}(m, \sigma, \mathrm{pk}_{sig}, \mathrm{pk}_{san}, \pi)$$

We require the usual correctness requirements to hold [9].

Experiment $\mathsf{SUnforgeability}^{\mathcal{SSS}}_{\mathcal{A}}(\lambda)$
$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
$(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^\lambda)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}}, \cdot, \cdot), \mathsf{Sanit}(\cdot, \cdot, \cdot, \cdot, sk_{\text{san}}), \mathsf{Proof}(sk_{\text{sig}}, \cdot, \cdot, \cdot, \cdot)}(pk_{\text{sig}}, pk_{\text{san}})$
for $i = 1, 2, \ldots, q$ let $(m_i, pk_{\text{san},i}, \text{ADM}_i)$ and σ_i index the queries/answers to/from Sign
for $j = 1, 2, \ldots, q'$ let $(m_j, \sigma_j, pk_{\text{sig},j}, \text{MOD}_j)$ and (m'_j, σ'_j) index the queries/answers to/from Sanit
if $\mathsf{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk_{\text{san}}) = \texttt{true} \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (pk_{\text{san}}, m^*, \underline{\sigma^*}) \neq (pk_{\text{san},i}, m_i, \underline{\sigma_i}) \wedge$
$\forall j \in \{1, 2, \ldots, q'\} : (pk_{\text{sig}}, m^*, \underline{\sigma^*}) \neq (pk_{\text{sig},j}, m'_j, \underline{\sigma'_j})$
return 1
return 0

Fig. 1. Strong Unforgeability

3 Revisiting the Security Properties

Here, we introduce the stronger security framework for $\mathcal{SSS}$s. The non-strong definitions are compiled from [9,11,12,13]. Our strong definitions are also based on the work done in [9,11,12,13]. We do not restate the properties we strengthen; we will highlight the differences in each definition. If one wants to also consider ADM as a part which needs to be protected, one needs to add the alterations presented in [26].

Next, we discuss the shortcomings of some of the security definitions and provide strictly stronger variants. In particular, we review and revise the following properties and introduce their strengthened/clarified definitions:

- Unforgeability
- Privacy
- Transparency
- Signer Accountability
- Sanitizer Accountability
- Non-Interactive Public Accountability

In a nutshell, for almost all of the aforementioned security definitions, we also consider the signature in the security definitions. As already argued, this seems to be a small technical detail, but has a significant impact on the resulting security. For transparency, which we think has the largest impact, we clarify this statement by introducing a (contrived) scheme which is trivially insecure in our setting, while it achieves the weaker state-of-the-art transparency notion.

3.1 Security of Sanitizable Signatures Re-Revisited

Strong Unforgeability. The first notion we want to revisit is unforgeability. The original definition given in [9] allows the adversary to derive new signatures on already queried messages, as the winning condition of the security game requires that the adversary outputs a message that was not queried before.

As already argued, this is not desired in all use-cases, very similar to the case of standard digital signatures. Hence, we alter the definition in such a way that the adversary cannot even generate any new signatures on its own, highlighted by the additional underlined conditions. In other words, in the strong definition of unforgeability, the adversary also wins if it can generate *any* new valid signature σ^* on its own.

Definition 2 (Strong Unforgeability). *An $\mathcal{SSS}$ is strongly unforgeable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\Pr[\mathit{SUnforgeability}^{\mathcal{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)\,,$$

where the experiment is defined in Figure 1.

We now obtain the following separation result:

Theorem 1. *Every scheme which is strongly unforgeable, is also unforgeable. The converse is not true.*

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

Fig. 2. Immutability

Proof. The proof is straightforward. Let $\mathcal{A}$ be an adversary winning the standard unforgeability game. We can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the strong unforgeability of an $\mathcal{SSS}$. $\mathcal{B}$ proceeds as follows:

1. $\mathcal{B}$ receives both public keys of its own challenger.
2. $\mathcal{B}$ passes the public keys to $\mathcal{A}$.
3. $\mathcal{B}$ simulates all oracles of $\mathcal{A}$ using its own oracles without any modifications.
4. Eventually, $\mathcal{A}$ outputs (m^*, σ^*). $\mathcal{B}$ also outputs (m^*, σ^*) as its own forgery.
5. As for all queries i to the signing oracle $(pk_{\text{san}}, m^*) \neq (pk_{\text{san},i}, m_i)$ yields and for all queries j to the sanitization oracle we also have $(pk_{\text{sig}}, m^*) \neq (pk_{\text{sig},j}, m'_j)$, (m^*, σ^*) clearly breaks the strong unforgeability of $\mathcal{SSS}$ with the same probability as $\mathcal{A}$ wins its own game.

The other direction is also easy: for each signature generation, we append a 0. For processing a signature, the last bit is removed. An adversary can exchange the 0 with a 1, while the signature verification is not affected. □

Jumping back to our example in §1, the new definition now prohibits the attacks: the accountant is bound to the signatures it receives from the server and cannot generate any new ones.

Immutability. As already aforementioned, a sanitizer should only be able to alter admissible blocks defined by ADM. Hence, also deleting or appending blocks must be prohibited. As usual, the adversary is given full oracle access, while it is also allowed to generate the sanitizer key pair.

Definition 3 (Immutability). *An $\mathcal{SSS}$ is immutable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that:*

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

where the experiment is defined in Figure 2.

Privacy. Privacy is related to the indistinguishability of ciphertexts. In particular, the adversary is allowed to input two messages with the same ADM which are sanitized to the exact same message. Then, the adversary has to decide which message was used to generate the sanitized one. Again, the adversary receives full adaptive oracle access.

Definition 4 (Privacy). *An $\mathcal{SSS}$ is private, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\left|\Pr[\mathit{Privacy}_{\mathcal{A}}^{\mathcal{SSS}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)\,,$$

where the experiment is defined in Figure 3.

Discussion. Compared to the original definition given in [9], we additionally check whether ADM "matches" *both* messages. This has a severe impact, depending on how one interprets the constraints given in [9]. Namely, [9] only requires that MOD_0 and MOD_1 are compatible with ADM, while "the resulting modified messages are identical for both tuples" [9]. However, it is not clear what this concretely means if ADM does not match one of the *messages*. More precisely, the adversary may proceed as follows: it chooses $m_0 = (1, 2)$ with $\text{ADM} = \{\{1, 2\}, \ell = 2\}$. Let the

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

Fig. 3. Privacy

Experiment $\mathsf{STransparency1}^{\mathcal{SSS}}_{\mathcal{A}}(\lambda)$
$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
$(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^\lambda)$
$b \leftarrow \{0,1\}$
$a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,sk_{\text{sig}},\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,sk_{\text{san}}),\mathsf{Proof}(sk_{\text{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{Sanit/Sign}(\cdot,\cdot,\cdot,sk_{\text{sig}},sk_{\text{san}},b)}(pk_{\text{sig}}, pk_{\text{san}})$
where oracle $\mathsf{Sanit/Sign}$ on input of $m, \text{MOD}, \text{ADM}$:
$\sigma \leftarrow \mathsf{Sign}(m, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})$
$(m', \sigma') \leftarrow \mathsf{Sanit}(m, \text{MOD}, \sigma, pk_{\text{sig}}, sk_{\text{san}})$
if $b = 1$:
$\sigma' \leftarrow \mathsf{Sign}(m', sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})$
return (m', σ')
if $\mathcal{A}$ has queried any $(m', \underline{\sigma'})$ output by $\mathsf{Sanit/Sign}$ to Proof, return a random bit
return 1, if $a = b$. Else, return 0.

Fig. 4. Strong Transparency V1

other message be $m_1 = (1)$. Clearly, $\text{ADM}(m_1) = 0$, i.e., ADM is not valid w.r.t. the *message* m_1. Further consider $\text{MOD}_0 = \text{MOD}_1 = \{(2,3)\}$. Depending on how one implements the modification algorithm *in detail*, we may have $\text{MOD}_0(m_0) = \text{MOD}_1(m_1) = (1,3)$, as it is not specified whether MOD returns $\perp$ for $\text{MOD}_1(m_1)$. However, now Sign returns $\perp$ for $b = 1$, but a signature if $b = 0$. Clearly, the adversary can easily derive b from the information it sees. We assume that prior art, e.g., [9,11], implicitly requires that $\text{MOD}(m)$ returns $\perp$ if $\text{ADM}(\text{MOD}) = 0$. In our definition, we made this explicit.

Strong Transparency. As mentioned earlier, the standard transparency definition only guarantees that the identity of the accountable party of a message m remains anonymous as long as no signature on this message m has ever been opened, i.e., no proof π has ever been created for that m. That is, as soon as the issuer of a signature on a certain message has been revealed, all previous and future signatures on the same message, regardless whether they are freshly signed or sanitized, may potentially be de-anonymized as well. As already argued, this might have undesirable side effects in practice. Our strong transparency definition in Figure 4 prohibits such attacks. Considering our example, even if it comes to a dispute over a sanitized criminal record, all other records remain transparent.

Definition 5 (Strong Transparency). *An $\mathcal{SSS}$ is strongly proof-restricted transparent, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\left|\Pr[\mathit{STransparency1}^{\mathcal{SSS}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)\,,$$

where the experiment is defined in Figure 4.

Experiment $\mathsf{STransparency2}^{\mathcal{SSS}}_{\mathcal{A}}(\lambda)$
$(sk_{\text{sig}}, pk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
$(sk_{\text{san}}, pk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^\lambda)$
$b \leftarrow \{0,1\}$
$\mathcal{Q} \leftarrow \emptyset$
$a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,sk_{\text{sig}},\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,sk_{\text{san}}),\mathsf{Proof}'(sk_{\text{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{Sanit/Sign}(\cdot,\cdot,\cdot,sk_{\text{sig}},sk_{\text{san}},b)}(pk_{\text{sig}}, pk_{\text{san}})$
where oracle Proof' on input of $sk_{\text{sig}}, m, \sigma, \{(m_i,\sigma_i) \mid i \in \mathbb{N}\}, pk'_{\text{san}}$:
return $\bot$, if $pk'_{\text{san}} = pk_{\text{san}} \wedge$
$((m,\sigma) \in \mathcal{Q} \vee \mathcal{Q} \cap \{(m_i,\sigma_i)\} \neq \emptyset)$
return $\mathsf{Proof}(sk_{\text{sig}}, m, \sigma, \{(m_i,\sigma_i)\}, pk'_{\text{san}})$
where oracle $\mathsf{Sanit/Sign}$ on input of $m, \text{MOD}, \text{ADM}, sk_{\text{sig}}, sk_{\text{san}}, b$:
$\sigma \leftarrow \mathsf{Sign}(m, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})$
$(m', \sigma') \leftarrow \mathsf{Sanit}(m, \text{MOD}, \sigma, pk_{\text{sig}}, sk_{\text{san}})$
if $b = 1$:
$\sigma' \leftarrow \mathsf{Sign}(m', sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})$
If $\sigma' \neq \bot$, set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m',\sigma')\}$
return (m', σ')
return 1, if $a = b$
return 0

Fig. 5. Strong Transparency V2

Remark. In the previous experiment, depicted in Fig. 4, the adversary does not win if it queried any output of the Sanit/Sign-oracle to the Proof-oracle. This definition may lead to some misconception. For example, in the original version of the papers by *Beck* et al. [4] and *Camenisch* et al. [14], i.e., the eprint-versions from 05/12/2017 [5], and 01/11/2017 [15], resp., use a slightly rephrased definition of transparency. Namely, they log the output of the Sanit/Sign-oracle, while they prohibit that the adversary inputs any message/signature pair as the *first* two arguments to the Proof-oracle (See Fig. 6). In this definition, the adversary still wins even if it queried a output of Sanit/Sign-oracle to the Proof-oracle, as long as this query was made before the Sanit/Sign-oracle returned the signature in question. This accounts for the insights gained by *Bellare* et al. [6].

However, defining it this way may lead to a trivial attack, leaking the bit directly. Consider the following strategy of the adversary:

1. Query the Sanit/Sign-oracle with an arbitrary message m with a single admissible block, receiving a signature σ.
2. Sanitize the message to a different message $m' \neq m$, receiving a signature σ'.
3. Query the Proof-oracle with $(m', \sigma', \{(m,\sigma)\}, pk_{\text{san}})$, receiving a proof π (which may be $\bot$).
4. If $\pi = \bot$, return 0.
5. If $\pi \neq \bot$, return 1.

Why does this strategy work? If the Sanit/Sign-oracle always freshly signs, the correctness definition requires that this signature, i.e., σ can be used to find the accountable party for the signature σ'. If, however, this is not the case, we directly know that the signature σ is the result of a sanitization. Note, due to the definition of correctness by *Brzuska* et al. [9], our attack *always* works.

Thus, as later fixed by *Beck* et al. [4] (eprint-version from 12/13/2017 [5]), it is important to also check the other argument of the Proof-oracle in some cases. We restate the fixed, and more intuitive definition, in Fig. 5.

After all, there may be schemes which do not require any non-sanitized signatures to find the accountable parties, e.g., schemes following the construction paradigm introduced by *Brzuska* et al. [11] (See §4), as they completely ignore the additional signatures during proof generation. Thus, there may be schemes which fulfill this, even stronger, version of transparency, as our attack presented above no longer applies. However, one must alter the syntax and correctness definition to make "Strong Transparency V3" a meaningful definition. We leave the construction and correctness of such schemes as an interesting research opportunity.

For completeness, the mentioned definition of transparency is given in Fig. 6.

Experiment $\mathsf{STransparency3}^{\mathcal{SSS}}_{\mathcal{A}}(\lambda)$
$(sk_{\text{sig}}, pk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
$(sk_{\text{san}}, pk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^\lambda)$
$b \leftarrow \{0,1\}$
$\mathcal{Q} \leftarrow \emptyset$
$a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,sk_{\text{sig}},\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,sk_{\text{san}}),\mathsf{Proof}'(sk_{\text{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{Sanit/Sign}(\cdot,\cdot,\cdot,sk_{\text{sig}},sk_{\text{san}},b)}(pk_{\text{sig}}, pk_{\text{san}})$
where oracle Proof' on input of $sk_{\text{sig}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, pk'_{\text{san}}$:
return $\perp$, if $pk'_{\text{san}} = pk_{\text{san}} \wedge$
$\underline{(m, \sigma) \in \mathcal{Q}}$
return $\mathsf{Proof}(sk_{\text{sig}}, m, \sigma, \{(m_i, \sigma_i)\}, pk'_{\text{san}})$
where oracle $\mathsf{Sanit/Sign}$ on input of m, MOD, ADM, sk_{sig}, sk_{san}, b:
$\sigma \leftarrow \mathsf{Sign}(m, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})$
$(m', \sigma') \leftarrow \mathsf{Sanit}(m, \text{MOD}, \sigma, pk_{\text{sig}}, sk_{\text{san}})$
if $b = 1$:
$\sigma' \leftarrow \mathsf{Sign}(m', sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})$
If $\sigma' \neq \perp$, set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m', \sigma')\}$
return (m', σ')
return 1, if $a = b$
return 0

Fig. 6. Strong Transparency V3

Discussion. Even though a bit more subtle (as anonymity is only potentially affected per message), the difference between the original definition and Definition 4 is comparable to the difference between, e.g., CPA- and CCA-anonymity for group signature schemes [27]. For CPA-anonymity the anonymity of the issuer of a signature is only guaranteed as long as *no* signature has never been inspected, i.e., de-anonymized, but all signers might be at risk once a single signature was inspected. However, for CCA-anonymity privacy is guaranteed for each signature as long as this specific signature has not been de-anonymized.

To clarify this statement in the context of $\mathcal{SSS}$s, we derive a (contrived) construction which is transparent, but trivially does not meet our strong transparency definition. Let $\mathcal{E} = \{\mathsf{EKGen}, \mathsf{Enc}, \mathsf{Dec}\}$ be a secret-key CPA-secure encryption scheme. We require that the secret-key and message space of $\mathcal{E}$ contains $\{0,1\}^\lambda$. Moreover, let $\mathcal{E}' = \{\mathsf{EKGen}', \mathsf{Enc}', \mathsf{Dec}'\}$ denote a labeled public-key CCA2-secure encryption scheme. Let $\mathcal{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a pseudo-random function, and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a random oracle. We implicitly assume that the key-space of $\mathcal{E}$ is $\{0,1\}^\lambda$.

Let $\mathcal{SSS} = (\mathsf{KGen}_{\text{sig}}, \mathsf{KGen}_{\text{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ be a sanitizable signature scheme according to the already existing standard definitions. We now construct a contrived sanitizable signature scheme $\mathcal{SSS}'$ from $\mathcal{SSS}$ which still fulfills the standard definitions but not strong transparency. For brevity, we skip obvious checks.[4]

Construction 1 *We define* $\mathcal{SSS}' = (\mathsf{KGen}'_{sig}, \mathsf{KGen}'_{san}, \mathsf{Sign}', \mathsf{Sanit}', \mathsf{Verify}', \mathsf{Proof}', \mathsf{Judge}')$ *as follows:*

1. *Key Generation/Signer* (KGen'_{sig})*:*
   (a) *Create a key* $\kappa \leftarrow \{0,1\}^\lambda$ *for a* $\mathcal{PRF}$ *at random.*
   (b) *Return* $(\text{pk}_{sig}, \text{sk}'_{sig})$*, where* $\text{sk}'_{sig} = (\kappa, \text{sk}_{sig})$*, and* $(\text{pk}_{sig}, \text{sk}_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^\lambda)$ *is generated as in the original* $\mathcal{SSS}$
2. *Key Generation/Sanitizer* (KGen'_{san})*:*
   (a) *Generate an additional key pair of a labeled (CCA2-secure) public-key encryption scheme, i.e.,* $(\text{pk}', \text{sk}') \leftarrow \mathsf{EKGen}'(1^\lambda)$
   (b) *Return* $((\text{pk}_{san}, \text{pk}'), (\text{sk}_{san}, \text{sk}'))$*, where* $(\text{pk}_{san}, \text{sk}_{san}) \leftarrow \mathsf{KGen}_{san}(1^\lambda)$*, i.e., the same keys as in the original* $\mathcal{SSS}$
3. *Signing* (Sign')*:*
   (a) *Compute* $\sigma \leftarrow \mathsf{Sign}(m, \text{sk}_{sig}, \text{pk}_{san}, \text{ADM})$*. If* $\sigma = \perp$*, return* $\perp$

[4] In the original publication [32], we use an encryption scheme without labels. However, without the labels, we cannot simulate the decryption in the proof. Likewise, for the symmetric encryption scheme, IND-CPA is sufficient.

   (b) $k \leftarrow \mathcal{PRF}(\kappa, \mathcal{H}(\mathrm{pk}_{san}))$
   (c) $k' \leftarrow \mathcal{PRF}(k, \mathcal{H}(m))$
   (d) *Return* $\sigma' = (\sigma, c_1, c_2)$, *where* $c_1 \leftarrow \mathsf{Enc}(k', 0^\lambda)$ *and* $c_2 \leftarrow \mathsf{Enc}'(\mathrm{pk}', k, (\mathrm{pk}_{sig}, \mathrm{pk}_{san}))$
4. *Sanitizing (*Sanit'*):*
   (a) *Parse* σ *as* (σ', c_1, c_2)
   (b) *Let* $k \leftarrow \mathsf{Dec}'(\mathrm{sk}', c_2, (\mathrm{pk}_{sig}, \mathrm{pk}_{san}))$
   (c) *Calculate* $(m', \sigma'') \leftarrow \mathsf{Sanit}(m, \mathrm{MOD}, \sigma', \mathrm{pk}_{sig}, \mathrm{sk}_{san})$. *If* $\sigma'' = \bot$, *return* $\bot$
   (d) *Let* $k' \leftarrow \mathcal{PRF}(k, \mathcal{H}(m')))$, $c_1' \leftarrow \mathsf{Enc}(k', 1^\lambda)$, *and* $c_2' \leftarrow \mathsf{Enc}'(\mathrm{pk}', k, (\mathrm{pk}_{sig}, \mathrm{pk}_{san}))$
   (e) *Return* $(m', (\sigma'', c_1', c_2'))$
5. *Verification (*Verify'*):*
   (a) *Parse* σ *as* (σ', c_1, c_2)
   (b) *Return* $\mathsf{Verify}(m, \sigma', \mathrm{pk}_{sig}, \mathrm{pk}_{san})$
6. *Proof (*Proof'*):*
   (a) *Parse* σ *as* (σ', c_1, c_2) *and each* (m_i, σ_i) *as* $(m_i, (\sigma_i', c_{1,i}, c_{2,i}))$.
   (b) *Let* $\pi \leftarrow \mathsf{Proof}(\mathrm{sk}_{sig}, m, \sigma', \{(m_i, \sigma_i')\}, \mathrm{pk}_{san})$. *If* $\pi = \bot$, *return* $\bot$
   (c) *Let* $k \leftarrow \mathcal{PRF}(\kappa, \mathcal{H}(\mathrm{pk}_{san}))$, *and* $k' \leftarrow \mathcal{PRF}(k, \mathcal{H}(m))$
   (d) *Return* $\pi' = (\pi, k')$.
7. *Judge (*Judge'*):*
   (a) *Parse* π *as* (π', k') *and* σ *as* (σ', c_1, c_2)
   (b) *Return* $\mathsf{Judge}(m, \sigma', \mathrm{pk}_{sig}, \mathrm{pk}_{san}, \pi')$

Theorem 2. *If* $\mathcal{SSS}$ *is secure,* $\mathcal{SSS}'$ *is also secure.*

Proof. We prove each property on its own.

Correctness. Follows from the correctness of the underlying $\mathcal{SSS}$.

Unforgeability. Assume, towards contradiction, that $\mathcal{SSS}'$ is not unforgeable. We can then construct an adversary $\mathcal{B}$ which breaks the unforgeability of the underlying $\mathcal{SSS}$. The reduction $\mathcal{B}$ proceeds as follows. It receives pk_{sig} and pk_{san} from its own challenger, draws κ and pk (for the encryption scheme) honestly and passes pk, pk_{sig} and pk_{san} to $\mathcal{A}$ for initialization. For every signing query, $\mathcal{B}$ queries its own signing oracle to receive the inner signature σ'. c_1 and c_2 are generated honestly by $\mathcal{B}$ using κ. Likewise, for sanitization, $\mathcal{B}$ strips of c_1 and c_2 and queries its own sanitization oracle to receive σ''. Then c_1 and c_2 are updated honestly. The combined sanitized signature is returned to $\mathcal{A}$. Finally, the proof-oracle is simulated honestly, i.e., the inner proof π is obtained by $\mathcal{B}$ using the proof-oracle provided (by supplying the inner signatures). The key k' is calculated honestly. Eventually, $\mathcal{A}$ returns a signature $(\sigma'^*, c_1^*, c_2^*)$, along with a message m^*. As, by assumption, m^* was never returned by $\mathcal{B}$'s signing oracle, $\mathcal{B}$ can return (m^*, σ'^*) as its own forgery attempt. As $\mathcal{A}$'s environment is simulated perfectly, $\mathcal{B}$'s advantage equals the one of $\mathcal{A}$.

Immutability. Assume, towards contradiction, that $\mathcal{SSS}'$ is not immutable. We can then construct an adversary $\mathcal{B}$ which breaks the immutability of the underlying $\mathcal{SSS}$. The reduction $\mathcal{B}$ proceeds as follows. It receives pk_{sig} from its own challenger, draws κ honestly and passes pk_{sig} to $\mathcal{A}$ for initialization. For every signing query, $\mathcal{B}$ queries its own signing oracle to receive the inner signature σ'. c_1 and c_2 are generated honestly by $\mathcal{B}$ using κ. The proof-oracle is simulated honestly, i.e., the inner proof π is obtained by $\mathcal{B}$ using the proof-oracle provided (by supplying the inner signatures). The key k' is calculated honestly. Eventually, $\mathcal{A}$ returns a tuple (m^*, σ^*, pk^*). As, by assumption, m^* is not derivable from the signatures seen, $\mathcal{B}$ can return (σ'^*, m^*, pk_1^*) (i.e., pk from the encryption scheme must be stripped off) as its own forgery attempt. As $\mathcal{A}$'s environment is simulated perfectly, $\mathcal{B}$'s advantage equals the one of $\mathcal{A}$.

Privacy. Assume, towards contradiction, that $\mathcal{SSS}'$ is not private. We can then construct an adversary $\mathcal{B}$ which breaks privacy of the underlying $\mathcal{SSS}$. To prove that our scheme is private, we use a sequence of games:

Game 0: The original privacy game.
Game 1: We now replace the generation of k using a purely random choice (which remains consistent with pk_{san}).
Transition - Game 0 → Game 1: Clearly, if this changes the view of the adversary, it breaks the pseudo-randomness of the $\mathcal{PRF}$. Thus, $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{prf}}(\lambda)$ follows.
Game 2: We now replace the content of each c_2 encrypted to pk with a 0.

Transition - Game 1 → Game 2: In this case, the adversary can break the encryption scheme used. The reduction works as follows. The reduction receives pk from the encryption scheme, generates all other values honestly (with the alterations from Game 1), and replaces each c_2 using a series of hybrids. Whatever the adversary $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. Note, the ciphertexts for all other pk_{san}s can still be decrypted due to the provided decryption oracle and different labels. Thus, $|\Pr[S_1] - \Pr[S_2]| \leq q\nu_{\mathsf{ind\text{-}cca2}}(\lambda)$ follows, where q is the number of generated c_2 with the challenge pk_{san}.

Game 3: We now abort, if the adversary guesses the bit b correctly.

Transition - Game 2 → Game 3: The only case left is that the underlying $\mathcal{SSS}$ is not private. The reduction is simple. $\mathcal{B}$ receives pk_{sig} and pk_{san} from its own challenger, pk (for the encryption scheme) honestly and passes pk, pk_{sig} and pk_{san} to $\mathcal{A}$ for initialization. Then, all oracles are simulated according to Game 2, querying the provided oracles to get each inner signature σ'. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{sss\text{-}privacy}}(\lambda)$ follows.

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

Transparency. Assume, towards contradiction, that $\mathcal{SSS}'$ is not transpart. We can then construct an adversary $\mathcal{B}$ which breaks transparency of the underlying $\mathcal{SSS}$. To prove that our scheme is transparent, we use a sequence of games:

Game 0: The original transparency game.

Game 1: We now replace the generation of k using a purely random choice (which remains consistent with pk_{san}).

Transition - Game 0 → Game 1: Clearly, if this changes the view of the adversary, it breaks the pseudo-randomness of the $\mathcal{PRF}$. Thus, $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{prf}}(\lambda)$ follows.

Game 2: We now replace the content of each c_2 encrypted to pk with a 0.

Transition - Game 1 → Game 2: In this case, the adversary can break the encryption scheme used. The reduction works as follows. The reduction receives pk from the encryption scheme, generates all other values honestly (with the alterations from Game 1), and replaces each c_2 using a series of hybrids. Whatever the adversary $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. Note, the ciphertexts for all other pk_{san}s can still be decrypted due to the provided decryption oracle and different labels. Thus, $|\Pr[S_1] - \Pr[S_2]| \leq q\nu_{\mathsf{ind\text{-}cca2}}(\lambda)$ follows, where q is the number of generated c_2 with the challenge pk_{san}.

Game 3: We now abort, if the adversary queries the proof-oracle with a valid message/signature pair (m^*, σ^*) pair for the challenge pk_{san}, while m^* was neither generated by the signing oracle nor the sanitizing oracle, or the LoRSanit oracle.

Transition - Game 2 → Game 3: Clearly, this tuple breaks the unforgeability of the $\mathcal{SSS}$, as m^*. The reduction is essentially the same as for standard unforgeability and therefore omitted. (To simulate the LoR-oracle, the reduction simply tosses a coin and uses the sign-oracle and sanitizing-oracle to mimic this oracle.) Thus, $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{sss\text{-}unf}}(\lambda)$ follows.

Game 4: We now replace the generation of k' using a purely random choice (which remains consistent with m).

Transition - Game 3 → Game 4: Clearly, if this changes the view of the adversary, it breaks the pseudo-randomness of the $\mathcal{PRF}$. Thus, $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{prf}}(\lambda)$ follows.

Game 5: We now abort, if a key k' was drawn twice (for different messages).

Transition - Game 4 → Game 5: Due to the birthday paradox, this can only happen with negligible probability. Thus, $|\Pr[S_4] - \Pr[S_5]| \leq q^2/2^\lambda$ follows, where q is the number of different messages queried for the challenge pk_{san}.

Game 6: We now replace each c_1 for a message which not queried to the proof oracle with a random message.

Transition - Game 5 → Game 6: Here, the reduction uses its own challenge oracle to receive the challenge ciphertext to embed. Using a series of hybrids, we replace each c_1 generated iteratively. Note, all other ciphertexts can be generated using the encryption oracle provided. Thus, $|\Pr[S_5] - \Pr[S_6]| \leq q\nu_{\mathsf{cpa}}(\lambda)$ follows, where q is the number of replaced ciphers.

Game 7: We now abort, if the adversary guesses the bit b correctly with a probability non-negligibly better than $\frac{1}{2}$.

Transition - Game 6 → Game 7: The only case left is that the underlying $\mathcal{SSS}$ is not transparent. The reduction is simple. $\mathcal{B}$ receives pk_{sig} and pk_{san} from its own challenger, pk (for the encryption scheme) honestly and passes pk, pk_{sig} and pk_{san} to $\mathcal{A}$ for initialization. Then, all oracles are simulated according to Game 3, querying the provided oracles to get each inner signature σ'. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_6] - \Pr[S_7]| \leq \nu_{\mathsf{sss\text{-}transparency}}(\lambda)$ follows.

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

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

Fig. 7. Unlinkability

Signer-Accountability. Essentially the same reduction as for unforgeability, but only the sanitization-oracle must be simulated, while the key pk^* returned by $\mathcal{A}$ is also returned.

Sanitizer-Accountability. Essentially the same reduction as for unforgeability, but the sanitization oracle must not be simulated, while the key "inner" pk_1^* returned by $\mathcal{A}$ is also returned.

Note, however, as Proof now also returns the secret key to decrypt c_1, an adversary can now decide for each *message* which has been queried to Proof how the corresponding signature was generated by decrypting, which is contained in the *signature* σ. We want to stress that [19] also defines transparency in a similar way. However, their scheme does not fulfill this definition, as the authors do not use strongly unforgeable signatures. In [12,13] transparency is defined similarly; however, as their schemes fulfill non-interactive public accountability and not transparency, this is not of importance.

Theorem 3. *Every scheme which is strongly transparent, is transparent. The converse is not true.*

Proof. The claim immediately follows from the construction above.

Unlinkability. Unlinkability prohibits an adversary to decide how a signature was generated, i.e., from which signature a sanitized signature was derived. We introduce the stronger definition from [13], where even the signer can be malicious. This game is similar to privacy with the same constraints. However, compared to the privacy game, the adversary can also input the signatures and the modification instruction. Again, it receives full oracle access; the signing and proof oracles can be simulated by the adversary.

Definition 6 (Unlinkability). *An $\mathcal{SSS}$ is unlinkable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\left|\Pr[\mathit{Unlinkability}_{\mathcal{A}}^{\mathcal{SSS}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)\,,$$

where the experiment is defined in Figure 7.

Next, we revise the three accountability definitions. As already argued, these definitions capture the intermediate cases, i.e., when either the sanitizer or the signer is adversarial. This is captured within the next three definitions.

Strong Signer-Accountability. For strong signer-accountability, a signer should not be able to blame a sanitizer if the sanitizer is actually not responsible for a given message/signature *pair* never generated by the sanitizer. Hence, the adversary has to generate a proof π^* which makes Judge to decide that the sanitizer is accountable, if it is not for a message/signature pair (m, σ). Here, the adversary gains access to all oracles related to sanitizing.

Definition 7 (Strong Signer Accountability). *An $\mathcal{SSS}$ is strongly signer accountable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\Pr[\mathit{SSig\text{-}Accountability}_{\mathcal{A}}^{\mathcal{SSS}}(\lambda) = 1] \leq \nu(\lambda)\,,$$

where the experiment is defined in Figure 8.

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

Fig. 8. Strong Signer Accountability

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

Fig. 9. Strong Sanitizer Accountability

Theorem 4. *Every scheme which is strongly signer-accountable, is also signer-accountable. The converse is not true.*

Proof. Similar to the proof of Theorem 1.

Strong Sanitizer-Accountability. Strong sanitizer-accountability is similar to standard sanitizer-accountability. However, compared to the original definition, the adversary now also wins if it can generate any signature σ^* which has not been generated by the signer. As usual, the adversary gains access to all signer-related oracles.

Definition 8 (Strong Sanitizer-Accountability). *An $\mathcal{SSS}$ is sanitizer accountable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\Pr[\mathit{SSan\text{-}Accountability}^{\mathcal{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)\,,$$

where the experiment is defined in Figure 9.

Theorem 5. *Every scheme which is strongly sanitizer-accountable, is also sanitizer-accountable. The converse is not true.*

Proof. Similar to the proof of Theorem 1.

Strong Non-Interactive Public Accountability. Strong non-interactive public accountability is also similar to the standard definition of non-interactive public accountability. However, this definition now also covers the case where the adversary could generate new signatures.

Experiment $\mathsf{SPubaccountability}^{\mathcal{SSS}}_{\mathcal{A}}(\lambda)$
$(pk_{\text{sig}}, sk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^\lambda)$
$(pk_{\text{san}}, sk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^\lambda)$
$(pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{\text{sig}}, \cdot, \cdot), \mathsf{Sanit}(\cdot, \cdot, \cdot, \cdot, sk_{\text{san}})}(pk_{\text{sig}}, pk_{\text{san}})$
for $i = 1, 2, \ldots, q$ let $(m_i, \text{ADM}_i, pk_{\text{san},i})$ and σ_i index the queries/answers to/from Sign
for $j = 1, 2, \ldots, q'$ let $(m_j, \text{MOD}_j, \sigma_j, pk_{\text{sig},j})$ and (m'_j, σ'_j) index the queries/answers to/from Sanit
if $\mathsf{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk^*) = \texttt{true} \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (pk^*, m^*, \underline{\sigma^*}) \neq (pk_{\text{san},i}, m_i, \underline{\sigma_i}) \wedge$
$\mathsf{Judge}(m^*, \sigma^*, pk_{\text{sig}}, pk^*, \bot) = \texttt{Sig}$
return 1
if $\mathsf{Verify}(m^*, \sigma^*, pk^*, pk_{\text{san}}) = \texttt{true} \wedge$
$\forall j \in \{1, 2, \ldots, q'\} : (pk^*, m^*, \underline{\sigma^*}) \neq (pk_{\text{sig},j}, m'_j, \underline{\sigma'_j}) \wedge$
$\mathsf{Judge}(m^*, \sigma^*, pk^*, pk_{\text{san}}, \bot) = \texttt{San}$
return 1
return 0

Fig. 10. Strong Public Accountability

Definition 9 (Strong Non-Interactive Public Accountability). *An $\mathcal{SSS}$ is strongly non-interactive publicly accountable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\Pr[\mathit{SPubaccountability}^{\mathcal{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)\,,$$

where the experiment is defined in Figure 10.

Theorem 6. *Every scheme which is strongly non-interactive publicly accountable, is also non-interactive publicly accountable. The converse is not true.*

Proof. Similar to the proof of Theorem 1.

Discussion. Clearly, all three definitions above fix the problems given in the running example: the server cannot generate any signature which has not been endorsed by the medical doctor, and vice versa.

Moreover, some known implications of the existing security properties from [9,11,12,13] carry over to our setting. Namely, strong sanitizer-accountability and strong signer-accountability together imply strong unforgeability, while strong non-interactive public accountability implies both strong sanitizer-accountability and strong signer-accountability. As strong transparency implies transparency, it also implies (proof-restricted) privacy. The proofs for this statement are essentially the same as given for the non-strong ones [9,11,12,13] and are therefore omitted. We leave it as open work to prove if other implications and separations are still valid in our setting.

4 Sketched Constructions

We now sketch how to achieve our new notions. As (strong) non-interactive public accountability and (strong) transparency are mutually exclusive, we need two distinct constructions.

Construction with Strong Transparency. A good candidate for a construction with strong transparency (and unlinkability) is the construction given by *Brzuska* et al. [11], if slightly altered. The main idea of the construction given in [11] is to sign all non-admissible blocks, the sanitizer's and the group's public key, and ADM using a strongly unforgeable deterministic signature scheme $\mathcal{S}$. Finally, the complete message, and the signer's public key are signed using a group signature scheme $\mathcal{GS}$ [7,11]. All standard properties of the $\mathcal{SSS}$ follow in a straightforward manner from the strong unforgeability and the group signature's security properties.[5] To achieve our stronger definitions, one needs a

[5] Note, their scheme only achieves a weaker form of unlinkability; also the signer's key pair is generated honestly. The adversary gains oracle access to Sign and Proof [11].

stronger group signature scheme. Namely, the anonymity, non-frameability and traceability security definitions [7,11] of the group signature scheme need to be adjusted. In particular, as done in §3, we need alter the games in the following way:[6]

- For anonymity, the adversary must never submit a pair (m, σ) for signature σ received for m from the LoR-Sign oracle to the opening oracle (instead of only m).
- For non-frameability and traceability, the adversary must never have received σ^* for m^* (instead of only not asking for a signature on m^*) from the signing oracle.

We also require that all other information in the full signature is non-malleably attached to it, e.g., by also signing them.

We leave a formal proof of security of the sketched construction as open work.

Construction with Strong Non-Interactive Public Accountability. The construction given in [13] achieves non-interactive public accountability. The basic idea of their scheme is the same as in the aforementioned construction, but instead of using a group signature $\mathcal{GS}$, they use another deterministic and unique signature scheme $\mathcal{S}$. Due to the determinism and uniqueness of the signature schemes deployed, their scheme already fulfills our strong definitions without any modifications. It depends on the use-case if a deterministic $\mathcal{SSS}$ offers the required properties from a real-world perspective.

5 Conclusion

We have shown that the state-of-the-art security definitions of $\mathcal{SSS}$s are not sufficient for some use-cases we have identified. We introduced new strictly stronger definitions, accounting for these additional use-cases. In particular, we have introduced the notions of strong unforgeability, strong transparency, strong sanitizer-accountability, strong signer-accountability and strong non-interactive public accountability and clarified the property of privacy. Finally, we have sketched how to alter an existing construction to achieve our enhanced notions.

References

1. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, a. shelat, and B. Waters. Computing on authenticated data. *J. Cryptology*, 28(2):351–395, 2015.
2. J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *EuroCrypt*, pages 83–107, 2002.
3. G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS*, pages 159–177, 2005.
4. M. T. Beck, J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Practical strongly invisible and strongly accountable sanitizable signatures. In *ACISP, Part I*, pages 437–452, 2017.
5. M. T. Beck, J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Practical strongly invisible and strongly accountable sanitizable signatures. *IACR Cryptology ePrint Archive*, 2017:445, 2017.
6. M. Bellare, D. Hofheinz, and E. Kiltz. Subtleties in the definition of IND-CCA: when and how should challenge decryption be disallowed? *J. Cryptology*, 28(1):29–48, 2015.
7. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EuroCrypt*, pages 614–629, 2003.
8. M. Bellare and P. Rogaway. The exact security of digital signatures - how to sign with RSA and rabin. In *EuroCrypt*, pages 399–416, 1996.
9. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of sanitizable signatures revisited. In *PKC*, pages 317–336, 2009.
10. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Santizable signatures: How to partially delegate control for authenticated data. In *BIOSIG*, 2009.
11. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of sanitizable signatures. In *PKC*, pages 444–461, 2010.
12. C. Brzuska, H. C. Pöhls, and K. Samelin. Non-interactive public accountability for sanitizable signatures. In *EuroPKI*, pages 178–193, 2012.
13. C. Brzuska, H. C. Pöhls, and K. Samelin. Efficient and perfectly unlinkable sanitizable signatures without group signatures. In *EuroPKI*, pages 12–30, 2013.

[6] Due to space requirements, we assume the reader is familiar with the security definitions of group signatures. References [7,11] contain all required definitions.

14. J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures. In *PKC, Part II*, pages 152–182, 2017.
15. J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Chameleon-hashes with ephemeral trapdoors and applications to invisible sanitizable signatures. *IACR Cryptology ePrint Archive*, 2017:11, 2017.
16. S. Canard and A. Jambert. On extended sanitizable signature schemes. In *CT-RSA*, pages 179–194, 2010.
17. S. Canard, A. Jambert, and R. Lescuyer. Sanitizable signatures with several signers and sanitizers. In *AfricaCrypt*, pages 35–52, 2012.
18. S. Canard, F. Laguillaumie, and M. Milhau. Trapdoorsanitizable signatures and their application to content protection. In *ACNS*, pages 258–276, 2008.
19. 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.
20. D. Demirel, D. Derler, C. Hanser, H. C. Pöhls, D. Slamanig, and G. Traverso. PRISMACLOUD D4.4: Overview of Functional and Malleable Signature Schemes. Technical report, H2020 Prismacloud, www.prismacloud.eu, 2015.
21. D. Derler, C. Hanser, and D. Slamanig. Blank digital signatures: Optimization and practical experiences. In *Privacy and Identity Management for the Future Internet in the Age of Globalisation - 9th IFIP WG 9.2, 9.5, 9.6/11.7, 11.4, 11.6/SIG 9.2.2 International Summer School, Patras, Greece, September 7-12, 2014, Revised Selected Papers*, pages 201–215, 2014.
22. D. Derler and D. Slamanig. Rethinking privacy for extended sanitizable signatures and a black-box construction of strongly private schemes. In *ProvSec*, 2015.
23. V. Fehr and M. Fischlin. Sanitizable signcryption: Sanitization over encrypted data (full version). *IACR Cryptology ePrint Archive*, 2015:765, 2015.
24. N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schröder, and M. Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In *PKC Part-I*, pages 301–330, 2016.
25. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
26. J. Gong, H. Qian, and Y. Zhou. Fully-secure and practical sanitizable signatures. In *Inscrypt*, pages 300–317, 2010.
27. J. Groth. Fully anonymous group signatures without random oracles. In *AsiaCrypt*, pages 164–180, 2007.
28. C. Hanser and D. Slamanig. Blank digital signatures. In *AsiaCCS*, pages 95–106, 2013.
29. F. Höhne, H. C. Pöhls, and K. Samelin. Rechtsfolgen editierbarer signaturen. *Datenschutz und Datensicherheit*, 36(7):485–491, 2012.
30. Q. Huang, D. S. Wong, and Y. Zhao. Generic transformation to strongly unforgeable signatures. In *ACNS*, pages 1–17, 2007.
31. M. Klonowski and A. Lauks. Extended sanitizable signatures. In *ICISC*, pages 343–355, 2006.
32. S. Krenn, K. Samelin, and D. Sommer. Stronger security for sanitizable signatures. In *DPM/QASA*, pages 100–117, 2015.
33. M. Mambo, K. Usuda, and E. Okamoto. Proxy signatures for delegating signing operation. In *CCS*, pages 48–57, 1996.
34. K. Miyazaki, S. Susaki, M. Iwamura, T. Matsumoto, R. Sasaki, and H. Yoshiura. Digital documents sanitizing problem. Technical Report 195, Institute of Electronics, Information and Communication Engineers, 2003.
35. H. C. Pöhls, K. Samelin, and J. Posegga. Sanitizable signatures in XML signature - performance, mixing properties, and revisiting the property of transparency. In *ACNS*, pages 166–182, 2011.
36. D. H. Yum, J. W. Seo, and P. Joong Lee. Trapdoor sanitizable signatures made easy. In *ACNS*, pages 53–68, 2010.

A Security Definitions Building Blocks

A.1 Labeled Public-Key Encryption Schemes $\mathcal{E}'$

Correctness. For a public-key encryption scheme $\mathcal{E}'$ we require the correctness properties to hold. In particular, we require that for all $\lambda \in \mathbb{N}$, for all $(sk, pk) \leftarrow \mathsf{EKGen}'(1^\lambda)$, for all $m \in \mathcal{M}$, for all $\theta \in \{0,1\}^*$, we have $\mathsf{Dec}'(sk, \mathsf{Enc}'(pk, m, \theta), \theta) = m$. This definition captures perfect correctness.

IND-CCA2 Security. IND-CCA2 Security requires that an adversary $\mathcal{A}$ cannot decide which message is actually contained in a ciphertext c, while in the case of CCA2 $\mathcal{A}$ receives full adaptive access to the decryption oracle. We also require that the message space $\mathcal{M}$ implicitly defines an upper bound on the message length, i.e., $|m|$. In other words, this means that the length is implicitly hidden for all messages in $\mathcal{M}$. From a practical viewpoint, this can be implemented using suitable padding techniques.

Definition 10 (IND-CCA2 Security). *A labeled public-key encryption scheme $\mathcal{E}'$ is IND-CCA2 secure, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that $\left|\Pr[\mathsf{IND\text{-}CCA2}^{\mathcal{E}'}_{\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$. The corresponding experiment is depicted in Fig. 11.*

Experiment $\mathsf{IND\text{-}CCA2}^{\mathcal{E}'}_{\mathcal{A}}(\lambda)$
$(sk, pk) \leftarrow \mathsf{EKGen}'(1^\lambda)$
$b \leftarrow \{0,1\}$
$(m_0, m_1, \theta^*, \mathtt{state}_{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathcal{O}}(pk)$
where $\mathcal{O} \leftarrow \mathsf{Dec}'(sk, \cdot, \cdot)$.
if $m_0 \notin \mathcal{M} \ \vee \ m_1 \notin \mathcal{M}$, let $c \leftarrow \perp$
else, let $c \leftarrow \mathsf{Enc}'(pk, m_b, \theta^*)$
$a \leftarrow \mathcal{A}^{\mathcal{O}}(c, \mathtt{state}_{\mathcal{A}})$
where $\mathcal{O} \leftarrow \mathsf{Dec}'(sk, \cdot, \cdot)$.
$\mathsf{Dec}''(sk, \cdot)$ behaves as Dec', but returns $\perp$ if queried with (c, θ^*).
return 1, if $a = b$
return 0

Fig. 11. Labeled IND-CCA2 security

Experiment $\mathsf{IND\text{-}CPA}^{\mathcal{E}}_{\mathcal{A}}(\lambda)$
$sk \leftarrow \mathsf{EKGen}'(1^\lambda)$
$b \leftarrow \{0,1\}$
$(m_0, m_1, \mathtt{state}_{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathsf{Enc}(sk,\cdot)}(1^\lambda)$
if $m_0 \notin \mathcal{M} \ \vee \ m_1 \notin \mathcal{M}$, let $c \leftarrow \perp$
else, let $c \leftarrow \mathsf{Enc}(sk, m_b)$
$a \leftarrow \mathcal{A}^{\mathsf{Enc}(sk,\cdot)}(c, \mathtt{state}_{\mathcal{A}})$
return 1, if $a = b$
return 0

Fig. 12. IND-CPA security

A.2 Secret-Key Encryption Schemes $\mathcal{E}$

Correctness. For a secret-key encryption scheme $\mathcal{E}$ we require the correctness properties to hold. In particular, we require that for all $\lambda \in \mathbb{N}$, for all $sk \leftarrow \mathsf{EKGen}(1^\lambda)$, for all $m \in \mathcal{M}$ we have $\mathsf{Dec}(sk, \mathsf{Enc}(sk, m)) = m$. This definition captures perfect correctness.

IND-CPA Security. IND-CPA Security with requires that an adversary $\mathcal{A}$ cannot decide which message is actually contained in a ciphertext c, while in the case of CPA $\mathcal{A}$ does not receive access to a decryption oracle. We also require that the message space $\mathcal{M}$ implicitly defines an upper bound on the message length, i.e., $|m|$. In other words, this means that the length is implicitly hidden for all messages in $\mathcal{M}$. From a practical viewpoint, this can be implemented using suitable padding techniques.

Definition 11 (IND-CPA Security). *An encryption scheme $\mathcal{E}$ is IND-CPA secure, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that $\left|\Pr[\mathsf{IND\text{-}CPA}^{\mathcal{E}}_{\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$. The corresponding experiment is depicted in Fig. 12.*

A.3 Pseudo-Random Functions $\mathcal{PRF}$

Definition 12 (Pseudo-Random Functions ($\mathcal{PRF}$s)). *A pseudo-random function $\mathcal{PRF}$ consists of two algorithms (PRFKGen, PRFEval) such that:*

PRFKGen. *The algorithm PRFKGen on input security parameter λ outputs the secret key of the $\mathcal{PRF}$: $\kappa \leftarrow \mathsf{PRFKGen}(1^\lambda)$.*
PRFEval. *The algorithm PRFEval gets as input the key κ, and the value $x \in \{0,1\}^\lambda$ to evaluate. It outputs the evaluated value $v \leftarrow \mathsf{PRFEval}(\kappa, x)$, $v \in \{0,1\}^\lambda$.*

Experiment $\mathsf{Pseudo\text{-}Randomness}_{\mathcal{A}}^{\mathcal{PRF}}(\lambda)$
  $\kappa \leftarrow \mathsf{PRFKGen}(1^\lambda)$
  $b \leftarrow \{0,1\}$
  $f \leftarrow F_\lambda$
  $a \leftarrow \mathcal{A}^{\mathsf{PRFEval}'(\kappa,\cdot)}(1^\lambda)$
    where oracle $\mathsf{PRFEval}'$ on input κ, x:
      return $\perp$, if $x \notin \{0,1\}^\lambda$
      if $b = 0$, return $\mathsf{PRFEval}(\kappa, x)$
      return $f(x)$
  return 1, if $a = b$
  return 0

Fig. 13. Pseudo-Randomness

Pseudo-Randomness. We require that $\mathcal{PRF}$ is actually pseudo-random. In the definition, let $F_\lambda = \{f : \{0,1\}^\lambda \to \{0,1\}^\lambda\}$ be the set of all functions mapping a value $x \in \{0,1\}^\lambda$ to a value $v \in \{0,1\}^\lambda$.

Definition 13 (Pseudo-Randomness). *A pseudo-random function $\mathcal{PRF}$ is pseudo-random, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that* $\left|\Pr[\mathsf{Pseudo\text{-}Randomness}_{\mathcal{A}}^{\mathcal{PRF}}(1^\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 13.*