A preliminary version of this paper appears in the proceedings of the *16th International Conference on Applied Cryptography and Network Security (ACNS 2018)*, Leuven, Belgium, July 2-4, 2018, Proceedings. © Springer-Verlag 2018. This is the full version.

Invisible Sanitizable Signatures and Public-Key Encryption are Equivalent

Marc Fischlin Patrick Harasser(✉)

Cryptoplexity, Technische Universität Darmstadt, Germany
`www.cryptoplexity.de`
`{marc.fischlin, patrick.harasser}@cryptoplexity.de`

April 11, 2018

Abstract. Sanitizable signature schemes are signature schemes which support the delegation of modification rights. The signer can allow a sanitizer to perform a set of admissible operations on the original message and then to update the signature, in such a way that basic security properties like unforgeability or accountability are preserved. Recently, Camenisch *et al.* (PKC 2017) devised new schemes with the previously unattained invisibility property. This property says that the set of admissible operations for the sanitizer remains hidden from outsiders. Subsequently, Beck *et al.* (ACISP 2017) gave an even stronger version of this notion and constructions achieving it. Here we characterize the invisibility property in both forms by showing that invisible sanitizable signatures are equivalent to $\mathsf{IND-CPA}$-secure encryption schemes, and strongly invisible signatures are equivalent to $\mathsf{IND-CCA2}$-secure encryption schemes. The equivalence is established by proving that invisible (resp. strongly invisible) sanitizable signature schemes yield $\mathsf{IND-CPA}$-secure (resp. $\mathsf{IND-CCA2}$-secure) public-key encryption schemes and that, vice versa, we can build (strongly) invisible sanitizable signatures given a corresponding public-key encryption scheme.

Keywords. Sanitizable signatures · Invisibility · Public-key encryption · One-way functions · Digital signatures

Contents

1 Introduction

Sanitizable signature schemes enable the signer of a document to declare certain sections of the message as admissible for modification, so that another designated party (the sanitizer) can modify them and update the signature without affecting the authenticity and integrity of the immutable parts. The main motivation is to balance out the verifier's wish to check authenticity of parts of the original document and the signer's desire for privacy of the sanitized data. The idea of sanitizable signature schemes dates back to a work by Ateniese *et al.* [ACdT05].

In [ACdT05], the authors introduced several security properties for sanitizable signature schemes. Besides unforgeability against outsiders, a desirable property is immutability, which demands that even a malicious sanitizer can only alter admissible parts. Privacy asks that one cannot reconstruct the original document given only the sanitized version and signature, and its strengthening called unlinkability [BFLS10] says that one cannot determine the origin to a sanitized document among several known possibilities. Signer and sanitizer accountability ensure that in case of a dispute the parties can give a convincing proof of who created a signature, the signer or the sanitizer. A less common property is transparency, which should hide who created a signature, in case neither of the parties helps to determine the originator—this stands in contrast to public accountability, where no additional help is required to determine who signed the document.

1.1 Invisible Sanitizable Signatures

Recently, Camenisch *et al.* [CDK+17] formalized the notion of invisibility of sanitizable signatures. This property, formerly called strong transparency in [ACdT05], should hide which modifications a sanitizer is allowed to perform. In previous constructions the description of admissible operations, denoted ADM, had usually been attached in clear to the signature. Gong *et al.* [GQZ11] were the first to argue that this information can be of value, and later Camenisch *et al.* showed that hiding it may be a desirable goal. They also revised the theoretical framework of sanitizable signatures in order to capture the invisibility property, and gave constructions achieving it based on a new type of chameleon hash functions with ephemeral trapdoors. Soon after, Beck *et al.* [BCD+17] further strengthened the notion of invisibility.

In its basic form, invisibility protects against leakage of ADM if the sanitizer public key is only used in connection with a single signer. In applications this means that the sanitizer must create a fresh key pair for each user. Strong invisibility, on the other hand, allows to use the same sanitizer key pair with multiple signers. Beck *et al.* use unique signatures, $\mathsf{IND-CCA2}$-secure encryption, and collision-resistant chameleon hash functions to achieve strong invisibility.

Technically, the difference between the two invisibility notions lies in the capabilities of an adversary trying to establish which of two potential operation sets, ADM_0 or ADM_1, has been encoded as admissible into the signature. Given a challenge signature, the adversary may query a sanitizing oracle on it as long as the requested modification does not allow to distinguish the two cases trivially (this happens e.g. if the modification is admissible for one of the two sets but not for the other). For the basic invisibility notion the adversary can ask for sanitizations only in connection with the public key $\mathsf{pk}_{\mathsf{Sig}}$ of the genuine signer. In the stronger notion, the adversary can also request sanitizations of messages signed with other, possibly maliciously chosen signer keys $\mathsf{pk}'_{\mathsf{Sig}}$.

1.2 Our Contributions

In this work we show that invisibile sanitizable signature schemes and public-key encryption schemes are equivalent. Our equivalence proof consists of four parts.

Invisibility implies $\mathsf{IND-CPA}$-secure encryption. Our first result is to show that an invisible sanitizable signature scheme yields an $\mathsf{IND-CPA}$-secure bit-encryption scheme. An invisible scheme hides the actual admissible operations for a signature; we can use this property to securely embed a message bit b by using one of two fixed and distinct admissible operation descriptions (ADM_0 or ADM_1) to build a signature σ under a fresh signer key pair. The ciphertext consists of the signature σ and the signer public key $\mathsf{pk_{Sig}}$. Invisibility now guarantees that no outsider is able to distinguish the two cases.

The trapdoor information for decryption is the sanitizer secret key; his public key acts as the public key of the encryption scheme. With his secret key, the sanitizer can run the sanitization process and check via a distinguishing modification which operation ADM_b has been embedded: Only the admissible one (ADM_b) will result in a valid new signature. For the other operation (ADM_{1-b}), the modification should fail by the immutability property of the sanitizable scheme. Note that we obviously need some other security property besides invisibility, because it is easy to devise invisible signature schemes without any other security property, e.g. with constant signatures.

Strong invisibility implies $\mathsf{IND-CCA2}$-secure encryption. While the construction of an $\mathsf{IND-CPA}$-secure scheme via the embedding of the hidden ADM may be expected, we argue next that the same construction yields an $\mathsf{IND-CCA2}$-secure encryption scheme if the underlying sanitizable signature scheme is strongly invisible. This result is less conventional, since it links the sanitization for different signer keys with the ability to securely decrypt different ciphertexts.

The proof idea is to note that ciphertexts in our encryption system are of the form $(\sigma, \mathsf{pk_{Sig}})$. Given a challenge ciphertext $(\sigma, \mathsf{pk_{Sig}})$, recall that for $\mathsf{IND-CCA2}$-security we must allow for oracle decryptions of ciphertexts $(\sigma', \mathsf{pk}'_{\mathsf{Sig}}) \neq (\sigma, \mathsf{pk_{Sig}})$. Since decryption is performed via sanitization, and strong invisibility allows us to call the sanitizer for different keys $\mathsf{pk}'_{\mathsf{Sig}}$, we can easily decrypt ciphertexts of the form $(\sigma', \mathsf{pk}'_{\mathsf{Sig}})$ with $\mathsf{pk}'_{\mathsf{Sig}} \neq \mathsf{pk_{Sig}}$. To handle ciphertexts $(\sigma', \mathsf{pk_{Sig}})$ under the original signer key we rely on the strong unforgeability property of the signture scheme: it says that one cannot create fresh signatures σ' under $\mathsf{pk_{Sig}}$, and therefore an $\mathsf{IND-CCA2}$-adversary cannot submit valid oracle queries of this form.

In a sense, this result warrants the deployment of an $\mathsf{IND-CCA2}$-secure encryption scheme in the strongly invisible construction of [BCD+17]: any strongly invisible sanitizable signature scheme already implies $\mathsf{IND-CCA2}$-secure encryption systems. Note that we construct an $\mathsf{IND-CCA2}$-secure *bit* encryption scheme, but this in turn implies $\mathsf{IND-CCA2}$-secure *string* encryption [CMTV15, HLW12, MH15, Ms09].

$\mathsf{IND-CPA}$-secure encryption implies invisibility. Next we establish the converse implication, i.e. from $\mathsf{IND-CPA}$-secure public-key encryption schemes to invisible sanitizable signatures. Note that the existence of the former primitive also implies the existence of one-way functions (the argument is identical to the one in [Rom90, Lemma 1]), and thus of secure digital signature schemes [NY89, Rom90], so that we can use this building block in our construction as well. Besides invisibility, the derived sanitizable signature scheme has all the common properties, like unforgeablility, immutability, privacy, and accountability.

The construction idea is to have the signer sign every message block of the message with a different, ephemeral key, and then to protect this tuple of signatures with an additional signature under his secret key. This "message" part of the signature ensures unforgeability, privacy, and accountability. To enable the sanitizer to modify the admissible blocks, the signer appends another "administrative" signature over the description ADM and the tuple of secret keys used to sign the admissible blocks, both encrypted under the sanitizer public encryption key to allow for invisibility. If some admissible block has to be modified, the sanitizer can retrieve the corresponding ephemeral key via decryption, change the message block and then update the relevant signatures in the "message" part. Immutability then follows from the unforgeability of the underlying digital signature scheme: It is ensured by the fact that the sanitizer only receives the signing keys for the blocks he is allowed to modify.

We stress here that our construction does not achieve some less common properties, in particular transparency and unlinkability, and that our security reduction is non-tight. On the other hand, we regard our work as being above all a feasibility result, so that tightness—even though desirable—should not be viewed as essential, and we believe that invisible, non-transparent sanitizable signatures can have interesting applications: One can envision scenarios where it should be impossible to learn which (if any) message blocks have the potential to be altered, but on the other hand it should be clear who signed the document (e.g., for legal and accountability reasons).

$\mathsf{IND-CCA2}$-secure encryption implies strong invisibility. The noteworthy property of the above construction is that $\mathsf{IND-CPA}$-security suffices to achieve (ordinary) invisibility. With a slight technical twist, we interestingly achieve strong invisibility if we now have an $\mathsf{IND-CCA2}$-secure encryption scheme: Namely, we include the signer public key in the encryption of ADM and the trapdoor information for the sanitzer. Hence, together with our converse construction of $\mathsf{IND-CCA2}$-secure encryption from strong invisibility, we also characterize this form of invisibility through public-key encryption.

In light of the strongly invisible construction of Beck *et al.* [BCD$^+$17], which besides an $\mathsf{IND-CCA2}$-secure encryption scheme also relies on signature schemes and collision-resistant chameleon hash functions, our solution shows that the former (together with a regular signature scheme) suffices. Of course, the solution by Beck *et al.* is significantly more efficient.

1.3 Related Work

As mentioned above, sanitizable signature schemes were introduced by Ateniese *et al.* in their foundational work [ACdT05]. The first, and to this date widely adopted security model describing this primitive is due to Brzuska *et al.* [BFF$^+$09], where the authors formalized the unforgeability, immutability, privacy, transparency, and accountability properties of a sanitizable signature scheme with game-based security definitions. Later on, Brzuska *et al.* added the important unlinkability property [BFLS10, BPS13], as well as non-interactive public accountability [BPS12, BPS13], to the picture of security notions—see Appendix C for all the definitions.

Subsequently, the formal framework introduced by Brzuska *et al.* in [BFF$^+$09] came under scrutiny by Gong *et al.* [GQZ11], who pointed out that sanitizable signatures formalized as above were vulnerable to so-called rights-forge attacks. Their solution was to introduce stronger versions of unforgeability, immutability and accountability, which also consider the admissible blocks in the security experiments. Even stronger variants of unforgeability, privacy, transparency, and accountability were later provided by Krenn *et al.* [KSS16], who decided to also track the signatures in the definitions (in much the same way as for regular signature schemes, when upgrading from "ordinary" to strong unforgeability). Finally, the invisibility property was formalized by Camenisch *et al.* [CDK$^+$17], following ideas already discussed in [ACdT05], and recently further strengthened by Beck *et al.* [BCD$^+$17].

The above literature deals with sanitizable signature schemes as they are intended here. On the other hand, we point out that there are many other primitives and extensions that are closely related to, but slightly different from sanitizable signature schemes as treated in this work. Among these there are redactable signatures [BBD$^+$10, dMPPS14, DPSS16, JMSW02], sanitizable signatures where sanitizer modifications are limited to certain values [CJ10, DS15, KL06, PSP11] or where the signer is allowed to add sanitizers after having signed the message [CLM08, YSL10], sanitizable signatures supporting a multi-signer, multi-sanitizer paradigm [BFLS09, BPS13, CJL12], or allowing for sanitization of signed, encrypted data [DHO16, FF15]. More generally, we note that this whole body of literature falls under the broad category of computing on authenticated data [BBD$^+$10, GGOT15, GOT15]. We refer to the extensive overviews of Ahn *et al.* [ABC$^+$12] and Demirel *et al.* [DDH$^+$15] for further information.

We conclude the related work overview by mentioning that our work also continues a line of research started in [BFLS09], where the authors showed that it is possible to construct a sanitizable signature scheme achieving unforgeability, immutability, privacy, and accountability only assuming that arbitrary secure signature schemes exist, i.e. only assuming that one-way functions exist. In this regard, and in light of known separation results of public-key cryptography and one-wayness [IR89], our work proves that the same does most likely not hold for (strongly) invisible sanitizable signature schemes.

1.4 Organization

We introduce standard preliminary notation in Section 2. Section 3 is dedicated to the definition of sanitizable signature schemes (and the corresponding specific notation), an overview of the correctness and security notions, and a thorough discussion of the invisibility property. In Section 4 we show how to construct a public-key bit-encryption scheme from an invisible sanitizable signature scheme, and we prove the corresponding security results, whereas Section 5 is devoted to the converse implication. Finally, we draw some conclusions in Section 6.

2 Notational Preliminaries

In this work, $\mathbb{N} = \mathbb{Z}_{>0}$ denotes the set of strictly positive integers, and $\lambda \in \mathbb{N}$ will always denote the security parameter of a given cryptographic scheme (in unary notation, it is denoted by 1^λ).
A generic polynomial function $p : \mathbb{N} \to \mathbb{R}$ will be denoted by poly. A function $\mu : \mathbb{N} \to \mathbb{R}$ is called *negligible* if, for every positive polynomial function poly, there exists an integer $N_{\mathsf{poly}} \in \mathbb{N}$ such that $\mu(\lambda) < 1/\mathsf{poly}(\lambda)$ for every $\lambda > N_{\mathsf{poly}}$. A generic negligible function $\mu : \mathbb{N} \to \mathbb{R}$ will be denoted by negl, and the notation $\mu(\lambda) = \mathsf{negl}(\lambda)$ will be used to specify that μ is a negligible function.

We adopt throughout the standard model of computation, where algorithms (and all cryptographic parties involved) are modeled as Turing machines. An algorithm is called *efficient* if it corresponds to a probabilistic Turing machine running in time polynomial in the length of its input. For this definition to be meaningful, all algorithms will have to take 1^λ as an additional input.

If $\mathcal{A}$ is a probabilistic algorithm accepting x as an input, we write $a \leftarrow_{\$} \mathcal{A}(x)$ if a is assigned the output of algorithm $\mathcal{A}$ on input x and randomly chosen randomness r. If $\mathcal{A}$ is a deterministic algorithm accepting x as an input, we will use the notation $a \leftarrow \mathcal{A}(x)$ (instead of $a \leftarrow_{\$} \mathcal{A}(x)$) to point out that there is no randomness involved in the computation. Finally, the arrow $\leftarrow$ will also be used for assignment statements.

For every algorithm $\mathcal{A}$, we assume that there is a dedicated error symbol $\perp$ which is not part of the input or the output space of $\mathcal{A}$. Furthermore, we implicitly assume that $\mathcal{A}$ returns $\perp$ if one of its inputs already is equal to $\perp$.

3 Definition of Sanitizable Signatures

3.1 Notation

The starting point of our theoretical discussion on sanitizable signatures is the security model introduced by Brzuska *et al.* in [BFF$^+$09]. However, since invisibility will play a crucial role in our work, their framework has to be slightly adapted. Their approach often relies on the fact that the description ADM of admissible parts is recoverable from signatures, in direct contrast to the invisibility property which aims to hide this information. Thus, before we can actually start with the definition of sanitizable signatures, we need to introduce some preliminary notation. In doing so we mainly follow the work of Camenisch *et al.* [CDK$^+$17].

Messages $m \in \mathcal{M}$ are assumed to consist of a finite number of *blocks*, each block being an element from a set $\mathcal{B}$ (usually $\mathcal{B} \subseteq \{0,1\}^*$). The message space $\mathcal{M}$ is thus a subset of $\mathcal{B}^*$. We use the notation $m[i]$ to refer to the i-th block and write $m = (m[1], \dots, m[\ell])$ to stress that the message m consists of ℓ blocks.

Admissible blocks in a message $m = (m[1], \dots, m[\ell]) \in \mathcal{M}$ are identified by means of the parameter $\mathrm{ADM} = (A, l) \in \mathcal{P}(\mathbb{N}) \times \mathbb{N}$ (also called *sanitizing rights*), where $l \in \mathbb{N}$ denotes the total number of blocks a message must have, while $A \coloneqq \{a_1, \dots, a_j\}$ is the set containing the indices of the blocks the sanitizer is allowed to modify. Of course, here we need $1 \leq a_1, \dots, a_j \leq l$, a condition that we will always assume to be satisfied. We then say that ADM *matches* m if $\ell = l$, in which case we write $\mathrm{ADM}(m) = \top$ (otherwise $\mathrm{ADM}(m) = \bot$). If $\mathrm{ADM}_0 = (A_0, l)$ and $\mathrm{ADM}_1 = (A_1, l)$ are two sanitizing rights matching m, we define $\mathrm{ADM}_0 \cap \mathrm{ADM}_1 \coloneqq (A_0 \cap A_1, l)$. Similarly, to identify admissible block indices, we write $a \in \mathrm{ADM} = (A, l)$ if $1 \leq a \leq l$ and $a \in A$.

If $m = (m[1], \dots, m[\ell]) \in \mathcal{M}$, the actual modifications to certain blocks made by the sanitizer (i.e., the *sanitizing instructions*) are identified by means of the parameter $\mathrm{MOD} = (M, l) \in \mathcal{P}(\mathbb{N} \times \mathcal{B}) \times \mathbb{N}$, where $l \in \mathbb{N}$ denotes the total number of blocks in a message and $M \coloneqq \{(i_1, \bar{m}_1), \dots, (i_k, \bar{m}_k)\}$ denotes the set of changes made by the sanitizer. Here $(i, \bar{m}) \in M$ is intended to mean that the sanitizer will replace block $m[i]$ with $\bar{m}$. Again, here we need $1 \leq i_1, \dots, i_k \leq l$, which we will assume throughout. We then say that MOD *matches* m if $\ell = l$, in which case we write $\mathrm{MOD}(m)$ for the message m' obtained by modifying m according to MOD, i.e. $m' = \mathrm{MOD}(m)$ if and only if $m' = (m'[1], \dots m'[\ell]) \in \mathcal{M}$ and, for every $1 \leq i \leq \ell$,

$$m'[i] = \begin{cases} \bar{m}_i & \text{if } i \in \{i_1, \dots, i_k\} \\ m[i] & \text{otherwise.} \end{cases}$$

We write $\mathrm{MOD}(m) = \bot$ if MOD does not match m.

Finally, recall that the sanitizer is supposed to modify only message blocks declared as admissible by the signer. In this regard, the following notation will be useful: If $\mathrm{ADM} = (A, l_{\mathrm{ADM}})$ and $\mathrm{MOD} = (M, l_{\mathrm{MOD}})$ are as above, we say that MOD *matches* (or *is valid w.r.t.*) ADM if $l_{\mathrm{ADM}} = l_{\mathrm{MOD}}$ and $\widetilde{M} \subseteq A$, where $\widetilde{M} \coloneqq \{i_1, \dots, i_k\}$ is the set of indices of the blocks which the sanitizer intends to modify (as specified by M). In this case we write $\mathrm{MOD}(\mathrm{ADM}) = \top$, otherwise $\mathrm{MOD}(\mathrm{ADM}) = \bot$.

3.2 Definition of Sanitizable Signature Schemes

With the notation introduced above we are now ready to define sanitizable signature schemes. The definition is based on the one given by Brzuska *et al.* in [BFF+09] but takes into account that the sanitizing rights are no longer publicly recoverable from a valid message-signature pair. We remark here that, nonetheless, the sanitizer is always able to learn which message blocks he can modify by trying to sanitize them singularly and checking if the resulting signature is valid, an operation linear in the number of blocks of the message. This is the reason why we do not include ADM as an additional input to the Sanit algorithm: Either it is implicitly in the signatures or it must be communicated out-of-band.

Definition 3.1 *A* sanitizable signature scheme SSS *is a tuple of eight probabilistic polynomial-time algorithms* $\mathsf{SSS} \coloneqq (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *defined as follows:*

$\mathsf{pp} \leftarrow_{\$} \mathsf{PGen}(1^\lambda)$: *On input a security parameter $\lambda \in \mathbb{N}$, the algorithm* PGen *generates a tuple of public parameters of* SSS*. The tuple* pp *includes the security parameter 1^λ, the definition of the block set $\mathcal{B}$, the message space $\mathcal{M} \subseteq \mathcal{B}^*$, the space of signatures $\mathcal{S}_{\mathsf{Sig}}$, the space of sanitized signatures $\mathcal{S}_{\mathsf{San}}$, the signer key space $\mathcal{K}_{\mathsf{Sig}}$, and the sanitizer key space $\mathcal{K}_{\mathsf{San}}$ (together with two special symbols $\top, \bot \notin \mathcal{M} \cup \mathcal{S}_{\mathsf{Sig}} \cup \mathcal{S}_{\mathsf{San}} \cup \mathcal{K}_{\mathsf{Sig}} \cup \mathcal{K}_{\mathsf{San}} \cup \{0,1\}^* \cup \{\mathsf{Sig}, \mathsf{San}\}$), along with any other information needed to sign or sanitize messages, or to verify signatures, except for identities and user key pairs. For brevity, we define $\mathcal{S} \coloneqq \mathcal{S}_{\mathsf{Sig}} \cup \mathcal{S}_{\mathsf{San}}$ and $\mathcal{K} \coloneqq \mathcal{K}_{\mathsf{Sig}} \cup \mathcal{K}_{\mathsf{San}}$.*

$(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$: *On input a tuple of public parameters* pp*, the algorithm* $\mathsf{KGen}_{\mathsf{Sig}}$ *returns a signer key pair or an error message, that is* $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \in \mathcal{K}_{\mathsf{Sig}} \cup \{\perp\}$*. Here,* $\mathsf{pk}_{\mathsf{Sig}}$ *and* $\mathsf{sk}_{\mathsf{Sig}}$ *are the signer public and secret keys, respectively. We write* $\mathcal{K}_{\mathsf{Sig,pk}}$ *and* $\mathcal{K}_{\mathsf{Sig,sk}}$ *for the sets of all possible signer public and secret keys.*

$(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$: *On input a tuple of public parameters* pp*, the algorithm* $\mathsf{KGen}_{\mathsf{San}}$ *returns a sanitizer key pair or an error message, that is* $(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \in \mathcal{K}_{\mathsf{San}} \cup \{\perp\}$*. Here,* $\mathsf{pk}_{\mathsf{San}}$ *and* $\mathsf{sk}_{\mathsf{San}}$ *are the sanitizer public and secret keys, respectively. We write* $\mathcal{K}_{\mathsf{San,pk}}$ *and* $\mathcal{K}_{\mathsf{San,sk}}$ *for the sets of all possible sanitizer public and secret keys.*

$\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM})$: *On input a tuple of public parameters* pp*, a message* $m \in \mathcal{M}$*, signer secret and public keys* $\mathsf{sk}_{\mathsf{Sig}} \in \mathcal{K}_{\mathsf{Sig,sk}}$*,* $\mathsf{pk}_{\mathsf{Sig}} \in \mathcal{K}_{\mathsf{Sig,pk}}$*, a sanitizer public key* $\mathsf{pk}_{\mathsf{San}} \in \mathcal{K}_{\mathsf{San,pk}}$*, and sanitizing rights* $\mathrm{ADM} \in \mathcal{P}(\mathbb{N}) \times \mathbb{N}$*, the algorithm* Sign *returns a signature or an error message, that is* $\sigma \in \mathcal{S}_{\mathsf{Sig}} \cup \{\perp\}$*. We do* not *assume* ADM *to be publicly recoverable from the message* m *and a valid signature* $\sigma \neq \perp$*.*

$\sigma' \leftarrow_{\$} \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD})$: *On input a tuple of public parameters* pp*, a message* $m \in \mathcal{M}$*, a signature* $\sigma \in \mathcal{S}$*, sanitizer secret and public keys* $\mathsf{sk}_{\mathsf{San}} \in \mathcal{K}_{\mathsf{San,sk}}$*,* $\mathsf{pk}_{\mathsf{San}} \in \mathcal{K}_{\mathsf{San,pk}}$*, a signer public key* $\mathsf{pk}_{\mathsf{Sig}} \in \mathcal{K}_{\mathsf{Sig,pk}}$*, and modification instructions* $\mathrm{MOD} \in \mathcal{P}(\mathbb{N} \times \mathcal{B}) \times \mathbb{N}$*, the algorithm* Sanit *returns a sanitized signature or an error message, that is* $\sigma' \in \mathcal{S}_{\mathsf{San}} \cup \{\perp\}$*.*

$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m, \sigma, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$: *On input a tuple of public parameters* pp*, a message* $m \in \mathcal{M}$*, a signature* $\sigma \in \mathcal{S}$*, a signer public key* $\mathsf{pk}_{\mathsf{Sig}} \in \mathcal{K}_{\mathsf{Sig,pk}}$*, and a sanitizer public key* $\mathsf{pk}_{\mathsf{San}} \in \mathcal{K}_{\mathsf{San,pk}}$*, the deterministic algorithm* Verify *returns a bit* $d \in \{\top, \perp\}$*.*

$\pi \leftarrow_{\$} \mathsf{Proof}(\mathsf{pp}, m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^{k}, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$: *On input a tuple of public parameters* pp*, a message* $m \in \mathcal{M}$*, a signature* $\sigma \in \mathcal{S}$*, a set of additional message-signature pairs* $\{(m_i, \sigma_i)\}_{i=1}^{k} \in (\mathcal{M} \times \mathcal{S})^*$ *with a polynomial (in the security parameter* λ*) number of elements, signer secret and public keys* $\mathsf{sk}_{\mathsf{Sig}} \in \mathcal{K}_{\mathsf{Sig,sk}}$*,* $\mathsf{pk}_{\mathsf{Sig}} \in \mathcal{K}_{\mathsf{Sig,pk}}$*, and a sanitizer public key* $\mathsf{pk}_{\mathsf{San}} \in \mathcal{K}_{\mathsf{San,pk}}$*, the algorithm* Proof *returns a bit string or an error message, that is* $\pi \in \{0,1\}^* \cup \{\perp\}$*.*

$d \leftarrow \mathsf{Judge}(\mathsf{pp}, m, \sigma, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \pi)$: *On input a tuple of public parameters* pp*, a message* $m \in \mathcal{M}$*, a signature* $\sigma \in \mathcal{S}$*, a signer public key* $\mathsf{pk}_{\mathsf{Sig}} \in \mathcal{K}_{\mathsf{Sig,pk}}$*, a sanitizer public key* $\mathsf{pk}_{\mathsf{San}} \in \mathcal{K}_{\mathsf{San,pk}}$*, and a bit string* $\pi \in \{0,1\}^*$*, the deterministic algorithm* Judge *returns a bit or an error message, that is* $d \in \{\mathsf{Sig}, \mathsf{San}\} \cup \{\perp\}$*.*

Remark. Observe that Definition 3.1 is purely syntactic in nature. We will always assume that the obvious checks needed to ensure a semantically correct execution of a sanitizable signature scheme SSS will be carried out directly by the algorithms defining SSS. For example, given all the appropriate inputs, we assume that the algorithm Sign itself checks if the tuple pp of public parameters is well-formed, if $m \in \mathcal{M}$, if the keys $\mathsf{sk}_{\mathsf{Sig}}$, $\mathsf{pk}_{\mathsf{Sig}}$, and $\mathsf{pk}_{\mathsf{San}}$ are compatible with pp, and if $\mathrm{ADM}(m) = \top$ before producing the signature, and that it proceeds accordingly (e.g., outputting $\perp$) should this condition not be satisfied. Likewise, we expect Sanit to also carry out all these basic tests, and furthermore to check if the message-signature pair it is given is valid and if $\mathrm{MOD}(m) \neq \perp$. Similar remarks hold for the other algorithms.

3.3 Correctness and Security Properties of Sanitizable Signature Schemes

We now turn to the definition of correctness and the statement of security properties of a sanitizable signature scheme SSS. As for correctness, we follow Brzuska *et al.* [BFF+09] and subsequent work and require that the following three properties hold. We give only an informal description here and refer to Appendix B for complete definitions, as adapted to our framework.

- *Signing Correctness*: Every time an honest signer signs a message $m \in \mathcal{M}$ with sanitizing rights matching m, he produces a valid signature $\sigma \neq \perp$ such that (m, σ) verifies under the corresponding public keys;

- *Sanitizing Correctness*: Every time the intended sanitizer honestly sanitizes a valid message-signature pair $(m, \sigma) \in \mathcal{M} \times \mathcal{S}$ with sanitizing instructions MOD matching the sanitizing rights given to him by the signer, he produces a valid signature $\sigma' \neq \perp$ such that $(\mathrm{MOD}(m), \sigma')$ verifies under the corresponding public keys;
- *Proof Correctness*: Every time an honest signer generates a proof regarding a valid message-signature pair, Judge identifies the correct accountable party.

Next we discuss the relevant security properties of a sanitizable signature scheme SSS. Most of these properties were introduced in their basic form by Brzuska *et al.* in [BFF+09] and later in [BFLS10, BPS12]. These definitions are included for completeness, but we will be mainly concerned with their "strong" counterparts as formalized by Krenn *et al.* in [KSS16] and later adopted by Camenisch *et al.* [CDK+17] and Beck *et al.* [BCD+17]. The definitions we adopt take into account that the sanitizing rights ADM (which are no longer assumed to be publicly recoverable from a valid message-signature pair) are an information which needs protection, as work by Gong *et al.* [GQZ11] has shown. In particular, by requiring a sanitizable signature scheme to satisfy the "strong" versions of the unforgeability, signer- and sanitizer-accountability properties, we mostly avoid so-called *rights forge attacks* as discussed in [GQZ11] (for immutability the matter is more delicate—see Appendix C for further discussions).
We again give only a brief and intuitive description of the security properties here and refer the interested reader to Appendix C for complete definitions and the corresponding security experiments. Only the notion of invisibility, central to our work, will be discussed here in detail.

- *Unforgeability*: No adversary should be able to produce a valid message-signature pair never seen before;
- *Immutability*: It should not be possible for a malicious sanitizer to violate the sanitizing rights given by the signer, i.e. the sanitizer should be able to modify only message blocks previously declared as admissible by the signer;
- *Privacy*: This is the equivalent of semantic security in the context of sanitizable signatures. Given a valid, sanitized message-signature pair, no adversary should be able to recover any information about the original content of the sanitized blocks;
- *Transparency*: Given a valid message-signature pair, no adversary should be able to determine whether it was the signer or the sanitizer who produced the signature;
- *Signer-Accountability*: A malicious signer should not be able to produce a valid message-signature pair $(m, \sigma) \in \mathcal{M} \times \mathcal{S}$ and a proof which induces Judge into erroneously blaming the sanitizer for (m, σ);
- *Sanitizer-Accountability*: A malicious sanitizer should not be able to produce a valid message-signature pair $(m', \sigma') \in \mathcal{M} \times \mathcal{S}$ such that legitimate proofs generated by the signer induce Judge into blaming the signer for (m', σ');
- *Unlinkability*: Given a valid message-signature pair $(m', \sigma') \in \mathcal{M} \times \mathcal{S}$ that has been sanitized, no adversary should be able to decide from which known pair $(m, \sigma) \in \mathcal{M} \times \mathcal{S}$ it originated from;
- *Non-Interactive Public Accountability*: The party accountable for a valid message-signature pair can be determined publicly, without the need of any further information. In particular, the Proof algorithm is trivial.

We remark here that there also exists a "blockwise" variant of non-interactive public accountability (see Brzuska *et al.* [BPS12]), where the party accountable for any specific message block can be publicly determined from a valid message-signature pair. We do not give a formal account of this notion here, as it requires the introduction of a new algorithm Detect and thus a different model of sanitizable signatures (again, see [BPS12]).

3.4 (Strong) Invisibility

Loosely speaking, a sanitizable signature scheme is *invisible* if, given a valid message-signature pair $(m, \sigma) \in \mathcal{M} \times \mathcal{S}$, no adversary is able to decide if any specific message block is admissible (i.e., can be modified by the sanitizer) or immutable. This property was first introduced by Ateniese *et al.* in their foundational work [ACdT05] under the name "strong transparency" (an expression later fallen into disuse, not to be confused with the notion of transparency defined in the literature). However, they did not provide any formal definition or construction achieving it. It was later abandoned by Brzuska *et al.* [BFF$^+$09] on the grounds that it appeared to be too strong. Indeed, since they worked under the assumption that ADM is always publicly recoverable from a valid signature $\sigma \neq \perp$ (in obvious conflict with the invisibility notion), it was in fact unachievable. Later on, the invisibility property was considered by Camenisch *et al.* [CDK$^+$17], who defined it formally and gave the first provably secure construction of an invisible sanitizable signature scheme. A stronger version of invisibility was later defined by Beck *et al.* in [BCD$^+$17], where the sanitizer may use his public key with multiple signers.

In the invisibility security experiment, a signer and a sanitizer key pair are generated and a bit $b \leftarrow_{\$} \{0, 1\}$ is chosen uniformly at random and kept secret. An adversary $\mathcal{A}$ is given access to an oracle $\mathcal{O}^{\mathsf{LoR}}$ which, on input a message and two sanitizing rights ADM_0, ADM_1, produces a signature σ (under the signer secret key and the sanitizer public key) making ADM_b admissible. In addition, $\mathcal{A}$ has adaptive access to restricted signing, sanitizing, and proof oracles.

We remark that, in the above experiment, a restricted signing oracle (with fixed sanitizer public key $\mathsf{pk}_{\mathsf{San}}$) can be simulated by querying $\mathcal{O}^{\mathsf{LoR}}$ with $\mathrm{ADM}_0 = \mathrm{ADM}_1$. Furthermore, for sanitization requests of any message-signature pair $(m, \sigma) \in \mathcal{M} \times \mathcal{S}$ with $\sigma \leftarrow_{\$} \mathcal{O}^{\mathsf{LoR}}(m, \mathrm{ADM}_0, \mathrm{ADM}_1)$, $\mathcal{A}$ must be limited to modifications matching $\mathrm{ADM}_0 \cap \mathrm{ADM}_1$ in order to avoid trivial attacks exposing b. This is why all queries to and answers from $\mathcal{O}^{\mathsf{LoR}}$, together with the allowed sanitizing rights $\mathrm{ADM}_0 \cap \mathrm{ADM}_1$, are recorded in a "whitelist" W: Whenever $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sanit}}$, the oracle goes through the list W of previously signed messages, to see which blocks the adversary is indeed allowed to modify. If the query is accepted, the whitelist has to be updated to also include the new (sanitized) message-signature pair, with the same sanitizing rights as the original pair (this has to be done because a sanitized message could be sanitized again). In the basic invisibility property the answers are only computed for the given key $\mathsf{pk}_{\mathsf{Sig}}$.

The adversary's goal is to guess b, i.e., to decide which set of blocks the oracle $\mathcal{O}^{\mathsf{LoR}}$ has made admissible. The scheme SSS is invisible if no efficient adversary as above succeeds in doing so with probability significantly greater than 1/2.

We observe that the definition of invisibility already has the flavor of the "strong" variant of the definitions given in Appendix C, in that we always keep track of the signatures in the whitelist W. On the other hand, the main drawback of this definition is that it is not possible to query the sanitization oracle for keys different from the challenge ones. As remarked by Beck *et al.* [BCD$^+$17], this may have undesirable consequences: $\mathcal{A}$ could pose as another signer and, as soon as he gets access to a sanitization oracle, could potentially learn the bit b. This assumption is quite plausible in many real-world use cases. It holds for example if the sanitizing party, here modeled by the oracle $\mathcal{O}^{\mathsf{Sanit}}$, uses the same key pair for both an honest and a corrupt signer.

To address these concerns (and to give a definition of invisibility that also protects against dishonest signers), one can allow queries to the sanitization oracle with public keys chosen by the adversary $\mathcal{A}$. This approach leads to the definition of strong invisibility. The main difference between the invisibility and the strong invisibility experiments is that the adversary is allowed oracle queries to $\tilde{\mathcal{O}}^{\mathsf{LoR}}$ and $\tilde{\mathcal{O}}^{\mathsf{Sanit}}$ with adversarially chosen public keys. A sanitizable signature scheme secure in this stronger sense does not suffer from the flaw mentioned above. As a side effect, the signing oracle derived from $\tilde{\mathcal{O}}^{\mathsf{LoR}}$ is no longer restricted. The formal definition of (strong) invisibility is given below.

Definition 3.2 ((Strong) Invisibility) *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme and* $\lambda \in \mathbb{N}$ *be a security parameter. Consider an efficient adversary* $\mathcal{A}$ *which:*

a) Takes as input a tuple of public parameters pp *corresponding to the security parameter* λ*, a signer public key* $\mathsf{pk}_{\mathsf{Sig}}$*, and a sanitizer public key* $\mathsf{pk}_{\mathsf{San}}$*;*

b) Has access to a restricted left-or-right signing oracle $\mathcal{O}^{\mathsf{LoR}}$ *(resp.* $\tilde{\mathcal{O}}^{\mathsf{LoR}}$*), a restricted sanitizing oracle* $\mathcal{O}^{\mathsf{Sanit}}$ *(resp.* $\tilde{\mathcal{O}}^{\mathsf{Sanit}}$*), and a proof oracle* $\mathcal{O}^{\mathsf{Proof}}$*;*

c) Returns a bit $b^* \in \{0,1\}$*.*

For every such adversary $\mathcal{A}$*, let* $\mathbf{Exp}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda)$ *with* $T = \mathsf{Inv}$ *(resp.* $T = \mathsf{SInv}$*) be the experiment defined*

```
$\mathbf{Exp}^{\mathsf{Inv}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:
$\mathsf{pp} \leftarrow_{\$} \mathsf{PGen}(1^{\lambda})$
$(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$
$(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$
$b \leftarrow_{\$} \{0,1\}$
$W \leftarrow \emptyset$
$b^* \leftarrow_{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{LoR}}, \mathcal{O}^{\mathsf{Sanit}}, \mathcal{O}^{\mathsf{Proof}}}(\mathsf{pp}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
if $b = b^*$ then
    return 1
return 0
```

```
$\mathcal{O}^{\mathsf{LoR}}(m, \mathrm{ADM}_0, \mathrm{ADM}_1)$:
if $\mathrm{ADM}_0(m) = \bot \ \vee \ \mathrm{ADM}_1(m) = \bot$ then
    return $\bot$
$\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM}_b)$
$\mathrm{ADM} \leftarrow \mathrm{ADM}_0 \cap \mathrm{ADM}_1$
$W \leftarrow W \cup \{(m, \sigma, \mathrm{ADM})\}$
return $\sigma$
```

```
$\mathcal{O}^{\mathsf{Sanit}}(m, \sigma, \mathrm{MOD})$:
if $(\exists \mathrm{ADM})((m, \sigma, \mathrm{ADM}) \in W \wedge$
  $\mathrm{MOD}(\mathrm{ADM}) = \top)$ then
    $m' \leftarrow \mathrm{MOD}(m)$
    $\sigma' \leftarrow_{\$} \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}},$
                  $\mathsf{pk}_{\mathsf{San}}, \mathrm{MOD})$
    $W \leftarrow W \cup \{(m', \sigma', \mathrm{ADM})\}$
    return $\sigma'$
return $\bot$
```

```
$\mathcal{O}^{\mathsf{Proof}}\Big(m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^{k}, \mathsf{pk}'_{\mathsf{San}}\Big)$:
$\pi \leftarrow_{\$} \mathsf{Proof}(\mathsf{pp}, m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^{k}, \mathsf{sk}_{\mathsf{Sig}},$
              $\mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}'_{\mathsf{San}})$
return $\pi$
```

```
$\mathbf{Exp}^{\mathsf{SInv}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:
$\mathsf{pp} \leftarrow_{\$} \mathsf{PGen}(1^{\lambda})$
$(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$
$(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$
$b \leftarrow_{\$} \{0,1\}$
$W \leftarrow \emptyset$
$b^* \leftarrow_{\$} \mathcal{A}^{\tilde{\mathcal{O}}^{\mathsf{LoR}}, \tilde{\mathcal{O}}^{\mathsf{Sanit}}, \mathcal{O}^{\mathsf{Proof}}}(\mathsf{pp}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
if $b = b^*$ then
    return 1
return 0
```

```
$\tilde{\mathcal{O}}^{\mathsf{LoR}}(m, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM}_0, \mathrm{ADM}_1)$:
if $\mathrm{ADM}_0(m) = \bot \ \vee \ \mathrm{ADM}_1(m) = \bot$ then
    return $\bot$
$\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM}_b)$
if $\mathsf{pk}'_{\mathsf{San}} = \mathsf{pk}_{\mathsf{San}}$ then
    $\mathrm{ADM} \leftarrow \mathrm{ADM}_0 \cap \mathrm{ADM}_1$
    $W \leftarrow W \cup \{(m, \sigma, \mathrm{ADM})\}$
else if $\mathrm{ADM}_0 \neq \mathrm{ADM}_1$ then
    return $\bot$
return $\sigma$
```

```
$\tilde{\mathcal{O}}^{\mathsf{Sanit}}\Big(m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \mathrm{MOD}\Big)$:
$m' \leftarrow \mathrm{MOD}(m)$
$\sigma' \leftarrow_{\$} \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}'_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD})$
if $\mathsf{pk}'_{\mathsf{Sig}} = \mathsf{pk}_{\mathsf{Sig}}$ then
  if $(\exists \mathrm{ADM})((m, \sigma, \mathrm{ADM}) \in W \wedge$
    $\mathrm{MOD}(\mathrm{ADM}) = \top)$ then
      $W \leftarrow W \cup \{(m', \sigma', \mathrm{ADM})\}$
      return $\sigma'$
  return $\bot$
return $\sigma'$
```

Figure 1: Invisibility (left) and strong invisibility (right)

on the left-hand side (resp. right-hand side) of Figure 1. We say that SSS *is* invisible *(resp.* strongly invisible*) if, for every efficient adversary $\mathcal{A}$ as above,* $\mathbf{Adv}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda) = \mathbb{P}\left[\mathbf{Exp}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda) = 1\right] = \mathsf{negl}(\lambda)$*, where the probability is taken over the random coins used by $\mathcal{A}$, as well as the random coins used in the experiment.*

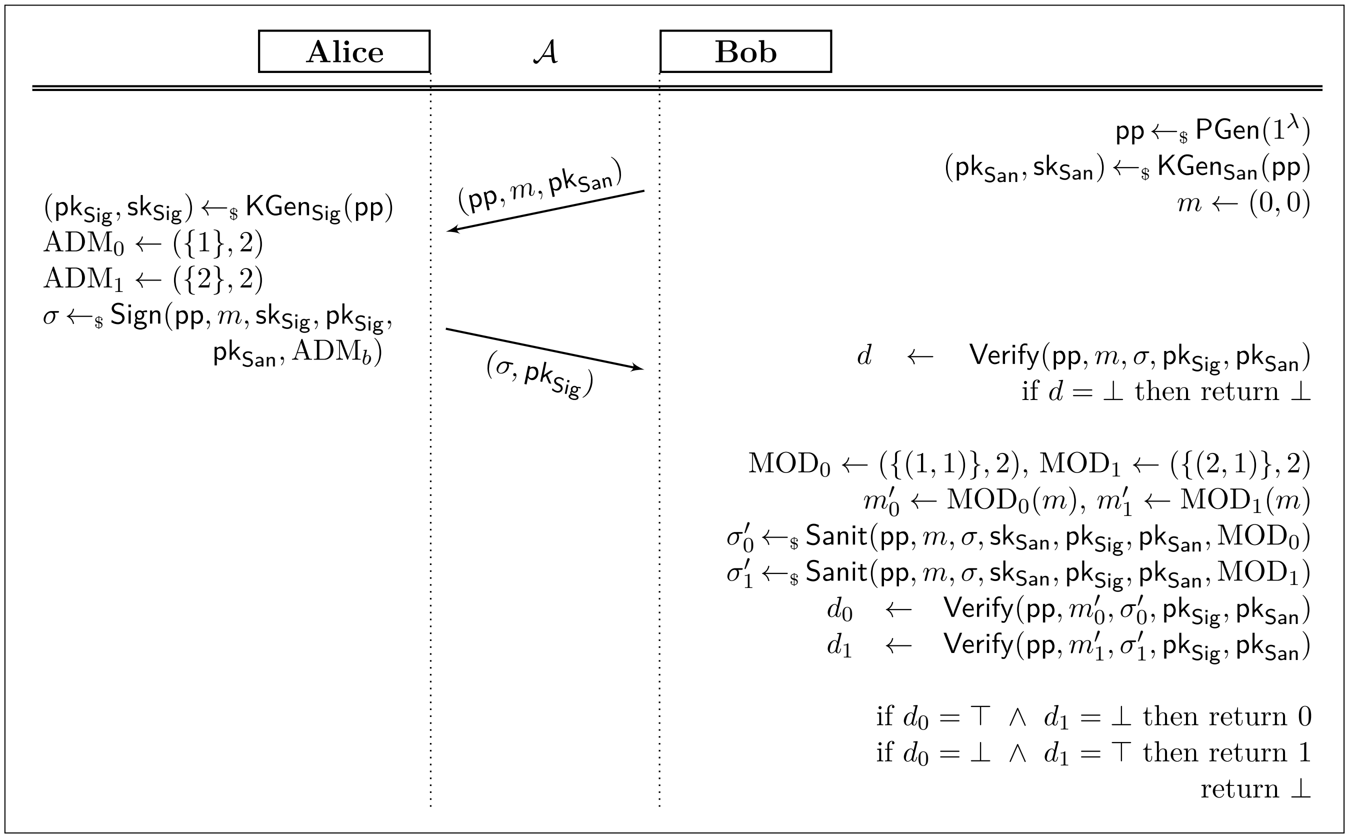
4 Invisible Sanitizable Signatures Imply Public-Key Encryption Schemes

In this section we show how to construct a public-key bit-encryption scheme from an invisible sanitizable signature scheme. We first give an informal overview of our construction and discuss how it is intended to be used. Afterwards, we define it more precisely and prove the correctness and security properties.

4.1 Construction

Suppose that Alice wants to send a secret bit $b \in \{0, 1\}$ to Bob, without an adversary $\mathcal{A}$ being able to learn it. To do so, they can use the protocol sketched in Figure 2 below. We present the main idea by viewing the scheme as a two-move key agreement protocol.

Bob publicly chooses a sanitizable signature scheme SSS and a security parameter $\lambda \in \mathbb{N}$, and generates a tuple of public parameters $\mathsf{pp} \leftarrow_{\$} \mathsf{PGen}(1^{\lambda})$. Observe that the block set $\mathcal{B}$ defined by pp clearly must contain at least two elements—we will assume that $\{0, 1\} \subseteq \mathcal{B}$, but for other block sets the adjustment is straightforward. Moreover, we assume that the two-block-messages $(0, 0), (1, 0), (0, 1)$ belong to the message space $\mathcal{M}$, but again our construction can be easily modified should this not be the case.


Alice | $\mathcal{A}$ | **Bob**

Bob:
$\mathsf{pp} \leftarrow_{\$} \mathsf{PGen}(1^{\lambda})$
$(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$
$m \leftarrow (0, 0)$

Bob → Alice: $(\mathsf{pp}, m, \mathsf{pk}_{\mathsf{San}})$

Alice:
$(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$
$\mathrm{ADM}_0 \leftarrow (\{1\}, 2)$
$\mathrm{ADM}_1 \leftarrow (\{2\}, 2)$
$\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM}_b)$

Alice → Bob: $(\sigma, \mathsf{pk}_{\mathsf{Sig}})$

Bob:
$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m, \sigma, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
if $d = \bot$ then return $\bot$

$\mathrm{MOD}_0 \leftarrow (\{(1, 1)\}, 2)$, $\mathrm{MOD}_1 \leftarrow (\{(2, 1)\}, 2)$
$m'_0 \leftarrow \mathrm{MOD}_0(m)$, $m'_1 \leftarrow \mathrm{MOD}_1(m)$
$\sigma'_0 \leftarrow_{\$} \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD}_0)$
$\sigma'_1 \leftarrow_{\$} \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD}_1)$
$d_0 \leftarrow \mathsf{Verify}(\mathsf{pp}, m'_0, \sigma'_0, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
$d_1 \leftarrow \mathsf{Verify}(\mathsf{pp}, m'_1, \sigma'_1, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$

if $d_0 = \top \wedge d_1 = \bot$ then return 0
if $d_0 = \bot \wedge d_1 = \top$ then return 1
return $\bot$

Figure 2: Intuitive construction of a public-key bit-encryption scheme from an invisible sanitizable signature scheme

Bob then generates a sanitizer key pair $(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$, and chooses a message $m \in \mathcal{M}$ consisting of two blocks, e.g. $m = (0, 0)$. He sends $(\mathsf{pp}, m, \mathsf{pk}_{\mathsf{San}})$ to Alice over an unprotected channel, while $\mathsf{sk}_{\mathsf{San}}$ is kept secret.

Upon receiving $(\mathsf{pp}, m, \mathsf{pk}_{\mathsf{San}})$, Alice runs $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$ to generate a signer key pair. Now, depending on whether she wants to encrypt $b = 0$ or $b = 1$, she signs the message m declaring as admissible the first or the second block, respectively. She then sends the signature σ and her public key $\mathsf{pk}_{\mathsf{Sig}}$ to Bob, while $\mathsf{sk}_{\mathsf{Sig}}$ is kept secret.

Upon receiving $(\sigma, \mathsf{pk}_{\mathsf{Sig}})$, Bob tries to separately modify the first and the second message block by replacing it with '1'. He thus sets $\mathrm{MOD}_0 \leftarrow (\{(1, 1)\}, 2)$ and $\mathrm{MOD}_1 \leftarrow (\{(2, 1)\}, 2)$ and then computes $\sigma_0' \leftarrow_{\$} \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD}_0)$ and $\sigma_1' \leftarrow_{\$} \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD}_1)$.

Now, assuming that SSS is sanitizing correct and immutable, exactly one of the two signatures computed by Bob will be valid. If Alice has encrypted $b = 0$, then σ_0' will be valid with overwhelming probability (because of the sanitizing correctness property), while σ_1' will be either invalid or equal to $\perp$ with very high probability (because SSS is immutable). On the other hand, if Alice has chosen $b = 1$, then σ_1' will be valid and σ_0' not by the same argument. In the unlikely event that both signatures are valid or both are invalid, Bob cannot decrypt the message sent by Alice.

We thus conclude that Bob is able to correctly decrypt the bit encrypted by Alice with very high probability by sanitizing m twice and checking the signatures (or error messages). Moreover, if we also assume SSS to be invisible, then any adversary $\mathcal{A}$ will be able to learn b only with negligible probability. In fact, from an outsider's perspective learning b is equivalent to establishing which message block is admissible, which is highly unlikely by the invisibility assumption.

We now turn to a more rigorous definition of our public-key bit-encryption scheme, as well as to the statement of the correctness and security properties.

Construction 4.1 *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme. We define a public-key bit-encryption scheme* Π *as in Figure 3.*

4.2 IND−CPA-Security

We now turn the informal argument that the construction yields an $\mathsf{IND{-}CPA}$-secure encryption scheme into a proof.

Theorem 4.2 *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme, and let* $\Pi := (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be the public-key bit-encryption scheme defined in Construction 4.1. If* SSS *is sanitizing correct, immutable and invisible, then* Π *is correct and* $\mathsf{IND{-}CPA}$*-secure.*

The proof gives a tight reduction in terms of the advantages: For any adversary $\mathcal{A}$ playing the $\mathsf{IND{-}CPA}$-game we construct an adversary $\mathcal{B}$ against the invisibility game with roughly the same running time as $\mathcal{A}$, such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{\mathsf{IND-CPA}}(\lambda) = \mathbf{Adv}_{\mathcal{B},\mathsf{SSS}}^{\mathsf{Inv}}(\lambda).$$

Note that we need the immutability property only to bound the correctness error by $2 \cdot \mathbf{Adv}_{\mathcal{C},\mathsf{SSS}}^{\mathsf{Imm}}(\lambda)$ for some efficient adversary $\mathcal{C}$ against the immutability game.

Proof. As for correctness, note that decryption of a genuinely encrypted bit b can only fail if sanitization with the correct modification MOD_b yields an invalid signature, or if the sanitization with the wrong

```
$\Pi.\mathsf{PGen}(1^\lambda)$:
$\mathsf{pp}_{\mathsf{SSS}} \leftarrow_\$ \mathsf{SSS.PGen}(1^\lambda)$
$\mathcal{M}_\Pi \leftarrow \{0,1\}$,
$\mathcal{C}_\Pi \leftarrow \mathcal{S}_{\mathsf{Sig}} \times \mathcal{K}_{\mathsf{Sig,pk}}$,
$\mathcal{K}_\Pi \leftarrow \mathcal{K}_{\mathsf{San}}$
$m \leftarrow (0,0)$
$\mathsf{pp}_\Pi \leftarrow (\mathsf{pp}_{\mathsf{SSS}}, \mathcal{M}_\Pi, \mathcal{C}_\Pi, \mathcal{K}_\Pi, m)$
return $\mathsf{pp}_\Pi$

$\Pi.\mathsf{KGen}(\mathsf{pp}_\Pi)$:
parse $\mathsf{pp}_\Pi = (\mathsf{pp}_{\mathsf{SSS}}, \mathcal{M}_\Pi, \mathcal{C}_\Pi, \mathcal{K}_\Pi, m)$
$(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_\$ \mathsf{SSS.KGen}_{\mathsf{San}}(\mathsf{pp}_{\mathsf{SSS}})$
$\mathsf{pk}_\Pi \leftarrow \mathsf{pk}_{\mathsf{San}},\ \mathsf{sk}_\Pi \leftarrow \mathsf{sk}_{\mathsf{San}}$
return $(\mathsf{pk}_\Pi, \mathsf{sk}_\Pi)$

$\Pi.\mathsf{Enc}(\mathsf{pp}_\Pi, b, \mathsf{pk}_\Pi)$:
parse $\mathsf{pp}_\Pi = (\mathsf{pp}_{\mathsf{SSS}}, \mathcal{M}_\Pi, \mathcal{C}_\Pi, \mathcal{K}_\Pi, m)$,
        $\mathsf{pk}_\Pi = \mathsf{pk}_{\mathsf{San}}$
$(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_\$ \mathsf{SSS.KGen}_{\mathsf{Sig}}(\mathsf{pp}_{\mathsf{SSS}})$
$\mathrm{ADM}_0 \leftarrow (\{1\}, 2)$
$\mathrm{ADM}_1 \leftarrow (\{2\}, 2)$
$\sigma \leftarrow_\$ \mathsf{SSS.Sign}(\mathsf{pp}_{\mathsf{SSS}}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}},$
        $\mathsf{pk}_{\mathsf{San}}, \mathrm{ADM}_b)$
return $(\sigma, \mathsf{pk}_{\mathsf{Sig}})$

$\Pi.\mathsf{Dec}(\mathsf{pp}_\Pi, c, \mathsf{pk}_\Pi, \mathsf{sk}_\Pi)$:
parse $\mathsf{pp}_\Pi = (\mathsf{pp}_{\mathsf{SSS}}, \mathcal{M}_\Pi, \mathcal{C}_\Pi, \mathcal{K}_\Pi, m)$, $c = (\sigma, \mathsf{pk}_{\mathsf{Sig}})$,
        $\mathsf{pk}_\Pi = \mathsf{pk}_{\mathsf{San}}$, $\mathsf{sk}_\Pi = \mathsf{sk}_{\mathsf{San}}$
$d \leftarrow \mathsf{SSS.Verify}(\mathsf{pp}_{\mathsf{SSS}}, m, \sigma, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
if $d = \perp$ then
    return $\perp$
$\mathrm{MOD}_0 \leftarrow (\{(1,1)\}, 2)$
$\mathrm{MOD}_1 \leftarrow (\{(2,1)\}, 2)$
$m'_0 \leftarrow \mathrm{MOD}_0(m)$,
$m'_1 \leftarrow \mathrm{MOD}_1(m)$
$\sigma'_0 \leftarrow_\$ \mathsf{SSS.Sanit}(\mathsf{pp}_{\mathsf{SSS}}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD}_0)$
$\sigma'_1 \leftarrow_\$ \mathsf{SSS.Sanit}(\mathsf{pp}_{\mathsf{SSS}}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD}_1)$
$d_0 \leftarrow \mathsf{SSS.Verify}(\mathsf{pp}_{\mathsf{SSS}}, m'_0, \sigma'_0, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
$d_1 \leftarrow \mathsf{SSS.Verify}(\mathsf{pp}_{\mathsf{SSS}}, m'_1, \sigma'_1, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
if $d_0 = \top \ \wedge \ d_1 = \perp$ then
    return 0
if $d_0 = \perp \ \wedge \ d_1 = \top$ then
    return 1
return $\perp$
```

Figure 3: Public-key bit-encryption scheme from an invisible sanitizable signature scheme

modification MOD_{1-b} yields a correct signature. The former can only happen with negligible probability by sanitizing correctness of SSS.

For the latter case we derive a contradiction to immutability of SSS. To see this, we consider both cases for $b \in \{0,1\}$, and for each choice construct an adversary $\mathcal{C}_b$ against immutability. Algorithm $\mathcal{C}_b$ receives as input $\mathsf{pp}_{\mathsf{SSS}}$ and a signer public key $\mathsf{pk}_{\mathsf{Sig}}$, and runs $\mathsf{KGen}_{\mathsf{San}}(1^\lambda)$ to create a sanitizer key pair $(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}})$. It picks the message $m = (0,0)$ and ADM_b as in Figure 3, and queries the signing oracle for a signature σ of $(m, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM}_b)$. Next it runs $\sigma' \leftarrow_\$ \mathsf{SSS.Sanit}(\mathsf{pp}_{\mathsf{SSS}}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD}_{1-b})$, and outputs the triple $(m, \sigma', \mathsf{pk}_{\mathsf{San}})$. Since the creation of the signature σ' coincides with decryption in Π, by assumption for at least one choice of b the adversary $\mathcal{C}_b$ will output a valid signature with non-negligible probability, and thus refute immutability.

As for security, assume SSS to be invisible; we show that then Π is $\mathsf{IND{-}CPA}$-secure. We argue security of the encryption scheme for a single challenge ciphertext only; the setting with multiple challenges follows from the simple case. Indeed, let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be any efficient two-stage adversary playing the $\mathsf{IND{-}CPA}$-game for Π, where $\mathcal{A}_1$ determines the message pair m_0, m_1 given the public encryption key, and $\mathcal{A}_2$ outputs a guess for b after having obtained the challenge ciphertext for m_b. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ playing the invisibility game for SSS, such that the probability of $\mathcal{A}$ winning the $\mathsf{IND{-}CPA}$-game and the probability of $\mathcal{B}$ winning the invisibility game coincide. By assumption, the latter probability is negligible and the theorem follows.

We let $\mathcal{B}$ play the invisibility game until right after the whitelist W has been set to $\emptyset$ (Line 5 of the invisibility experiment in Figure 1, where $\mathcal{B}$ receives $(\mathsf{pp}_{\mathsf{SSS}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$). Adversary $\mathcal{B}$ then runs $\mathcal{A}$ and gives $\mathcal{A}_1$ the parameters $\mathsf{pp}_\Pi = (\mathsf{pp}_{\mathsf{SSS}}, \mathcal{M}_\Pi, \mathcal{C}_\Pi, \mathcal{K}_\Pi, m)$ and $\mathsf{pk}_\Pi = \mathsf{pk}_{\mathsf{San}}$. Once $\mathcal{A}_1$ outputs (m_0, m_1, st), the two bit messages $m_0, m_1 \in \{0,1\}$ are passed on to $\mathcal{B}$, who then queries $\sigma^* \leftarrow_\$ \mathcal{O}^{\mathsf{LoR}}((0,0), \mathrm{ADM}_{m_0}, \mathrm{ADM}_{m_1})$ and returns the encryption $(\sigma^*, \mathsf{pk}_{\mathsf{Sig}})$ of m_b to $\mathcal{A}_2$ (together with the usual parameters and the state information st). Now, if b^* is the guess output by $\mathcal{A}$, let $\mathcal{B}$ also return b^*.

Observe that this procedure amounts to a genuine simulation of the indistinguishability game to $\mathcal{A}$, since $\mathcal{B}$'s answer is an authentic encryption of one of the two plaintexts. Also notice that $\mathcal{A}$ correctly guesses which plaintext has been encrypted (i.e., $\mathcal{A}$ wins the $\mathsf{IND{-}CPA}$-game) if and only if $\mathcal{B}$ correctly estimates which of the two message blocks has been declared as admissible (i.e., $\mathcal{B}$ wins the invisibility game). Hence, the probability of the two adversaries winning their respective games is the same, and this concludes the proof. $\square$

4.3 IND−CCA2-Security

We next argue that the scheme above achieves $\mathsf{IND{-}CCA2}$-security if SSS is assumed to be strongly invisibile. Recall that the difference to regular invisibility is that now the adversary against strong invisibility can also make left-or-right signature requests for $(m, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM}_0, \mathrm{ADM}_1)$ with different sanitizer public keys $\mathsf{pk}'_{\mathsf{San}} \neq \mathsf{pk}_{\mathsf{San}}$, and sanitization requests for $(m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \mathrm{MOD})$ with different signer public keys $\mathsf{pk}'_{\mathsf{Sig}} \neq \mathsf{pk}_{\mathsf{Sig}}$. Interestingly, for our construction and proof we only rely on the latter property.

For the security proof we also need strong unforgeability of the sanitizable signature scheme. The reason is that ciphertexts are of the form $(\sigma, \mathsf{pk}_{\mathsf{Sig}})$, and the $\mathsf{IND{-}CCA2}$-adversary may ask for decryptions of the form $(\sigma', \mathsf{pk}_{\mathsf{Sig}})$ where it alters the signature component for the same message. This would allow to break the security of the encryption scheme easily.

Theorem 4.3 *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme, and let* $\Pi := (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be the public-key bit-encryption scheme defined in Construction 4.1. If* SSS *is sanitizing correct, strongly unforgeable, immutable and strongly invisible, then* Π *is correct and* $\mathsf{IND{-}CCA2}$*-secure.*

The proof also gives a tight reduction in terms of the advantages: For any adversary $\mathcal{A}$ playing the $\mathsf{IND{-}CCA2}$-game we construct adversaries $\mathcal{B}$ and $\mathcal{C}$ with roughly the same running time as $\mathcal{A}$, such that

$$\mathbf{Adv}^{\mathsf{IND-CCA2}}_{\mathcal{A},\Pi}(\lambda) \leq \mathbf{Adv}^{\mathsf{SInv}}_{\mathcal{B},\mathsf{SSS}}(\lambda) + 2 \cdot \mathbf{Adv}^{\mathsf{SUnf}}_{\mathcal{C},\mathsf{SSS}}(\lambda).$$

In fact, for $\mathsf{IND{-}CCA1}$-security regular unforgeability is sufficient. Once more, we need immutability only to bound the correctness error.

Proof. Correctness of Π is proved as in Theorem 4.2. We now show that Π is $\mathsf{IND{-}CCA2}$-secure. We use the same strategy as above. Again, let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be any efficient two-stage adversary playing the $\mathsf{IND{-}CCA2}$-game for Π. We construct an efficient adversary $\mathcal{B}$ playing the strong invisibility game for SSS such that the probabilities of the two adversaries winning their respective games are negligibly close. Then, since by assumption $\mathcal{B}$ wins the strong invisibility game only with negligible probability, the same must hold for $\mathcal{A}$ playing the $\mathsf{IND{-}CCA2}$-game. Thus the theorem will follow from the strong invisibility assumption of SSS.

We construct $\mathcal{B}$ as we did in the proof of Theorem 4.2, with the only difference that now $\mathcal{B}$ also needs to be able to handle decryption queries requested by $\mathcal{A}$. This can be achieved using the oracle $\tilde{\mathcal{O}}^{\mathsf{Sanit}}$ that $\mathcal{B}$ has access to. Whenever $\mathcal{A}$ wishes to decrypt a ciphertext $(\sigma', \mathsf{pk}'_{\mathsf{Sig}})$, algorithm $\mathcal{B}$ proceeds as follows:

- If $\mathsf{pk}'_{\mathsf{Sig}} \neq \mathsf{pk}_{\mathsf{Sig}}$—note that $\mathcal{B}$ receives the public key $\mathsf{pk}_{\mathsf{Sig}}$ at the beginning of the invisibility game—then $\mathcal{B}$ runs $\Pi.\mathsf{Dec}$ as defined in Construction 4.1, with the two calls to $\mathsf{SSS.Sanit}$ substituted with corresponding calls to $\tilde{\mathcal{O}}^{\mathsf{Sanit}}$, and returns the answer to $\mathcal{A}$.
- If $\mathsf{pk}'_{\mathsf{Sig}} = \mathsf{pk}_{\mathsf{Sig}}$ then $\mathcal{B}$ immediately returns $\perp$.

Observe that this procedure amounts to a genuine simulation of the $\mathsf{IND-CCA2}$-game for $\mathcal{A}$, except potentially for the case where $\mathcal{A}$ queries ciphertexts $(\sigma', \mathsf{pk}'_{\mathsf{Sig}})$ with $\mathsf{pk}'_{\mathsf{Sig}} = \mathsf{pk}_{\mathsf{Sig}}$, the signer public key that $\mathcal{B}$ has been given at the beginning of the invisibility game it is playing. In such a case a genuine decryption oracle might indeed return a valid decryption for the ciphertext $(\sigma', \mathsf{pk}_{\mathsf{Sig}})$, whereas $\mathcal{B}$ would return $\perp$. There are two cases when such a valid decryption query $(\sigma', \mathsf{pk}_{\mathsf{Sig}})$ can happen: Either the query is made in the first phase, before $\mathcal{A}$ receives the challenge ciphertext, or it is made in the second phase—in which case, necessarily, $\sigma' \neq \sigma^*$, since $\mathcal{A}$ cannot query the decryption oracle on the challenge ciphertext. We argue below that the former case refutes unforgeability and the latter contradicts strong unforgeability of SSS.

Suppose that the probability of $\mathcal{A}$ making a valid query of the form $(\sigma', \mathsf{pk}_{\mathsf{Sig}})$ to its genuine decryption oracle before receiving the challenge ciphertext $(\sigma^*, \mathsf{pk}_{\mathsf{Sig}})$ were non-negligible. Then we can build an adversary $\mathcal{C}$ against the unforgeability of SSS as follows. Algorithm $\mathcal{C}$ initially receives $\mathsf{pp}_{\mathsf{SSS}}$ and keys $\mathsf{pk}_{\mathsf{Sig}}$, $\mathsf{pk}_{\mathsf{San}}$ as input and then runs $\mathcal{A}$'s attack, handing $\mathsf{pk}_{\mathsf{San}}$ to $\mathcal{A}$ but keeping $\mathsf{pk}_{\mathsf{Sig}}$ secret. Observe that $\mathcal{C}$ can simulate $\mathcal{A}$'s decryption oracle using it's own sanitization oracle as discussed above.

If at some point $\mathcal{A}$ makes a query of the form $(\sigma', \mathsf{pk}_{\mathsf{Sig}})$ to the decryption oracle which would not yield $\perp$, then the signature σ' must verify under the keys $\mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}$ for the message $m = (0, 0)$. Adversary $\mathcal{C}$ can verify the validity of the signature by himself and, if valid, abort $\mathcal{A}$'s simulation and output (m, σ') as his forgery. Since he has made neither any signature queries, nor any sanitization queries under $\mathsf{pk}_{\mathsf{Sig}}$ to his oracles yet, (m, σ') is indeed a successful forgery for a fresh message, thus leading to $\mathcal{C}$ winning the unforgeability experiment.

The second case, where $\mathcal{A}$'s query is made after receiving $(\sigma^*, \mathsf{pk}_{\mathsf{Sig}})$, is analogous. This time, however, a reduction $\mathcal{C}$ to the strong unforgeability needs to query its signing oracle to create the challenge ciphertext. But since $\mathcal{A}$'s request now involves a new signature $\sigma' \neq \sigma^*$, adversary $\mathcal{C}$ can still output (m, σ') for a successful break of the strong unforgeability property.

So we conclude that $\mathcal{B}$'s simulation is genuine, except for some negligible probability. It follows as before that a successful adversary $\mathcal{A}$ against the $\mathsf{IND-CCA2}$-secure encryption scheme yields a successful attacker $\mathcal{B}$ against strong invisibility. □

5 Public-Key Encryption Implies Invisible Sanitizable Signatures

In this section we present our construction of an invisible sanitizable signature scheme, starting from a secure public-key encryption scheme.

5.1 Construction

Our construction based on public-key encryption follows the established encode-and-sign paradigm and exploits the idea of using chameleon hash functions and signing the hash values with a regular signature scheme Σ (see, e.g., [ACdT05, BFF+09]). The sanitizer can then find collisions for the hashes with the help of his trapdoor key, allowing him to modify the message. Here, instead of chameleon hashes we use the signature scheme Σ itself.

In our scheme, signatures consist of two parts: the "message" part ensures the basic unforgeability and accountability properties, and can be created by either of the two parties. In contrast, the "administrative" part contains the information needed by the sanitizer to perform modifications, and can be created only by the signer. Parts of the administrative information are encrypted under an encryption scheme Π under the sanitizer's public key, to ensure invisibility.

To begin with, the signer generates a key pair $(\mathsf{pk}_\Sigma, \mathsf{sk}_\Sigma)$ for Σ, while the sanitizer creates keys $(\mathsf{pk}'_\Sigma, \mathsf{sk}'_\Sigma)$ and $(\mathsf{pk}_\Pi, \mathsf{sk}_\Pi)$ for Σ and Π, respectively. To sign a message $m = (m[1], \dots, m[\ell])$, the signer generates a new key pair $(\mathsf{pk}^i_\Sigma, \mathsf{sk}^i_\Sigma)$ $(1 \le i \le \ell)$ for each block of m, signs every block with the corresponding key, and creates a tuple of signatures $S \coloneqq (\sigma^1, \dots, \sigma^\ell)$. He then generates a signature σ_{MSG} of the message $(0, m, S, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$ under sk_Σ. Here, m and S are signed so that they are protected from modification by outsiders, whereas $\mathsf{pk}_{\mathsf{Sig}}$, $\mathsf{pk}_{\mathsf{San}}$ and the initial bit '0' are included for technical reasons (namely, domain separation). The "message" part of the final signature σ then consists of $(S, \sigma_{\mathrm{MSG}})$.

The first part of the signature must now be complemented with the information required to sanitize the admissible parts of the message, and to verify the signature. To this end, the signer generates the tuple $K_{\mathrm{ADM}} = (\mathsf{sk}^{i_1}_\Sigma, \mathsf{sk}^{i_2}_\Sigma, \dots)$ containing the secret keys of the admissible blocks $i_j \in \mathrm{ADM}$ (properly padded to ensure a length-invariant encoding), and encloses it for the sanitizing party via encryption under pk_Π. In addition, we also hide the parameter ADM (to ensure invisibility) and the signer public key (in foresight of the strongly invisible version of our result) in this encryption. In summary, the signer creates an encryption C of $(\mathsf{pk}_{\mathsf{San}}, K_{\mathrm{ADM}}, \mathrm{ADM})$ under pk_Π and then, in order to prevent changes in these administrative data, creates a (regular) signature σ_{ADM} of the message $(1, \mathsf{pk}_{\mathsf{San}}, V, C)$. Here, $V \coloneqq (\mathsf{pk}^1_\Sigma, \dots, \mathsf{pk}^l_\Sigma)$ contains the verification keys for the single blocks, and again the initial bit '1' is included for domain separation reasons. The "administrative" part of the signature is then $(V, C, \sigma_{\mathrm{ADM}})$, and the final signature is $\sigma \coloneqq (S, \sigma_{\mathrm{MSG}}, V, C, \sigma_{\mathrm{ADM}})$.

If the sanitizer receives a signature σ for a message m, he first checks the validity of the signatures S, σ_{MSG} and σ_{ADM}, and recovers ADM and the corresponding signing keys K_{ADM} by decrypting C. Then, given valid sanitizing instructions $m' = \mathrm{MOD}(m)$, he can update the "message" part of σ, leaving the "administrative" part unchanged. He obtains S' by substituting the relevant entries in S with new signatures of the modified blocks under the corresponding keys K_{ADM}, and updates σ'_{MSG} by re-signing $(0, m', S', \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$ under sk'_Σ. Finally, the sanitized signature for m' is given by $\sigma' = (S', \sigma'_{\mathrm{MSG}}, V, C, \sigma_{\mathrm{ADM}})$.

Immutability of the scheme is achieved by the fact that the sanitizer does not know the secret keys for the blocks he is not supposed to modify, and therefore cannot obtain suitable replacements for signatures in S. Observe that the signature σ_{MSG} immediately ensures public accountability, since it serves as a proof of who put the overall signature. This also implies that our scheme does not achieve transparency. For technical reasons it neither supports unlinkability.

Remark. The above discussion presumes that some mild assumptions on Σ and Π are satisfied, which we will henceforth assume to be in place. In particular, all signature keys must be of fixed length L (this can be achieved via padding of the keys), and the message blocks, as well as the tuples of the form $(0, m, S, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$ and $(1, \mathsf{pk}_{\mathsf{San}}, V, C)$, must lie in the message space of Σ (this is no restriction, because the signatures constructed in [NY89, Rom90] support messages of arbitrary polynomial length). Also, ADM must be encoded in a length-invariant manner, and tuples of the form $(\mathsf{pk}_{\mathsf{San}}, K_{\mathrm{ADM}}, \mathrm{ADM})$ must lie in the message space of Π (which can be achieved through hybrid encryption).

We now turn to a more rigorous definition of our sanitizable signature scheme, as well as to the statement of the correctness and security results.

Construction 5.1 *Let* $\Sigma \coloneqq (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a signature scheme and* $\Pi \coloneqq (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *a public-key encryption scheme. We define a sanitizable signature scheme* SSS *as in Figures 4 and 5 below.*

```
$\mathsf{SSS.PGen}(1^\lambda)$:
$\mathsf{pp}_\Pi \leftarrow_\$ \Pi.\mathsf{PGen}(1^\lambda)$
$\mathsf{pp}_\Sigma \leftarrow_\$ \Sigma.\mathsf{PGen}(1^\lambda)$
$\mathcal{M} \leftarrow \mathcal{M}_\Sigma^l$
$\mathsf{pp}_{\mathsf{SSS}} \leftarrow (\mathsf{pp}_\Pi, \mathsf{pp}_\Sigma, \mathcal{M})$
return $\mathsf{pp}_{\mathsf{SSS}}$

$\mathsf{SSS.KGen}_{\mathsf{Sig}}(\mathsf{pp}_{\mathsf{SSS}})$:
parse $\mathsf{pp}_{\mathsf{SSS}} = (\mathsf{pp}_\Pi, \mathsf{pp}_\Sigma, \mathcal{M})$
$(\mathsf{pk}_\Sigma, \mathsf{sk}_\Sigma) \leftarrow_\$ \Sigma.\mathsf{KGen}(\mathsf{pp}_\Sigma)$
$\mathsf{pk}_{\mathsf{Sig}} \leftarrow \mathsf{pk}_\Sigma$, $\mathsf{sk}_{\mathsf{Sig}} \leftarrow \mathsf{sk}_\Sigma$
return $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}})$

$\mathsf{SSS.KGen}_{\mathsf{San}}(\mathsf{pp}_{\mathsf{SSS}})$:
parse $\mathsf{pp}_{\mathsf{SSS}} = (\mathsf{pp}_\Pi, \mathsf{pp}_\Sigma, \mathcal{M})$
$(\mathsf{pk}_\Pi, \mathsf{sk}_\Pi) \leftarrow_\$ \Pi.\mathsf{KGen}(\mathsf{pp}_\Pi)$
$(\mathsf{pk}'_\Sigma, \mathsf{sk}'_\Sigma) \leftarrow_\$ \Sigma.\mathsf{KGen}(\mathsf{pp}_\Sigma)$
$\mathsf{pk}_{\mathsf{San}} \leftarrow (\mathsf{pk}_\Pi, \mathsf{pk}'_\Sigma)$
$\mathsf{sk}_{\mathsf{San}} \leftarrow (\mathsf{sk}_\Pi, \mathsf{sk}'_\Sigma)$
return $(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}})$
```

```
$\mathsf{SSS.Sign}(\mathsf{pp}_{\mathsf{SSS}}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM})$:
if $\mathrm{ADM}(m) = \perp$ then
    return $\perp$
parse $\mathsf{pp}_{\mathsf{SSS}} = (\mathsf{pp}_\Pi, \mathsf{pp}_\Sigma, \mathcal{M})$, $m = (m[1], \ldots, m[l])$,
    $\mathsf{sk}_{\mathsf{Sig}} = \mathsf{sk}_\Sigma$, $\mathsf{pk}_{\mathsf{Sig}} = \mathsf{pk}_\Sigma$, $\mathsf{pk}_{\mathsf{San}} = (\mathsf{pk}_\Pi, \mathsf{pk}'_\Sigma)$,
    $\mathrm{ADM} = (A, l)$
$V \leftarrow \emptyset$, $S \leftarrow \emptyset$, $K_{\mathrm{ADM}} \leftarrow \emptyset$
for $1 \leq i \leq l$ do
    $(\mathsf{pk}^i_\Sigma, \mathsf{sk}^i_\Sigma) \leftarrow_\$ \Sigma.\mathsf{KGen}(\mathsf{pp}_\Sigma)$
    $V \leftarrow V \cup \{(i, \mathsf{pk}^i_\Sigma)\}$
    $\sigma^i \leftarrow_\$ \Sigma.\mathsf{Sign}(\mathsf{pp}_\Sigma, m[i], \mathsf{sk}^i_\Sigma, \mathsf{pk}^i_\Sigma)$
    $S \leftarrow S \cup \{(i, \sigma^i)\}$
    $K_{\mathrm{ADM}} \leftarrow K_{\mathrm{ADM}} \cup \begin{cases} \{(i, \mathsf{sk}^i_\Sigma)\} & \text{if } i \in A \\ \{(i, 0^L)\} & \text{else} \end{cases}$
parse $V = (\mathsf{pk}^1_\Sigma, \ldots, \mathsf{pk}^l_\Sigma)$, $S = (\sigma^1, \ldots, \sigma^l)$,
    $K_{\mathrm{ADM}} = (K^1_{\mathrm{ADM}}, \ldots, K^l_{\mathrm{ADM}})$
$t \leftarrow (0, m, S, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
$\sigma_{\mathrm{MSG}} \leftarrow_\$ \Sigma.\mathsf{Sign}(\mathsf{pp}_\Sigma, t, \mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma)$
$C \leftarrow_\$ \Pi.\mathsf{Enc}(\mathsf{pp}_\Pi, (\mathsf{pk}_{\mathsf{Sig}}, K_{\mathrm{ADM}}, \mathrm{ADM}), \mathsf{pk}_\Pi)$
$u \leftarrow (1, \mathsf{pk}_{\mathsf{San}}, V, C)$
$\sigma_{\mathrm{ADM}} \leftarrow_\$ \Sigma.\mathsf{Sign}(\mathsf{pp}_\Sigma, u, \mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma)$
$\sigma \leftarrow (S, \sigma_{\mathrm{MSG}}, V, C, \sigma_{\mathrm{ADM}})$
return $\sigma$
```

Figure 4: Invisible sanitizable signature scheme from a public-key encryption scheme: parameter generation, signer and sanitizer key generation, and signing algorithms.

5.2 Security

The formal security statement for our construction is given in Theorem 5.2.

Theorem 5.2 *If the signature scheme Σ is correct and unforgeable, and the encryption scheme Π is correct, then the sanitizable signature scheme* SSS *in Construction 5.1 is correct. If Σ is unforgeable and Π is* IND−CPA*-secure, then* SSS *is unforgeable, immutable, private, publicly accountable, and invisible.*

We show the theorem by a sequence of lemmas, one for each property. Given that our Proof algorithm is trivial (in fact, we achieve public accountability, which means that Judge can decide based on direct inspection of the signature, without any additional help), we omit the Proof oracle in all proofs below.

Lemma 5.3 *If the signature scheme Σ is correct and unforgeable, and the encryption scheme Π is correct, then the sanitizable signature scheme* SSS *in Construction 5.1 is correct.*

Proof. Consider the definition of the verification algorithm in Figure 5. It shows that the only possibilities for Verify to return $\perp$ are either $d_1 = d_2 = \perp$, or $d_3 = \perp$, or that some signature σ^i $(1 \leq i \leq l)$ does not verify under pk^i_Σ. But for a genuinely generated signature σ, all these only happen with negligible probability by the correctness of Σ. This establishes signing correctness of SSS.

```
$\mathsf{SSS.Verify}(\mathsf{pp_{SSS}}, m, \sigma, \mathsf{pk_{Sig}}, \mathsf{pk_{San}})$:
parse $\mathsf{pp_{SSS}} = (\mathsf{pp}_\Pi, \mathsf{pp}_\Sigma, \mathcal{M})$,
        $m = (m[1], \dots, m[l])$,
        $\sigma = (S, \sigma_{\mathrm{MSG}}, V, C, \sigma_{\mathrm{ADM}})$,
        $S = (\sigma^1, \dots, \sigma^l)$,
        $V = (\mathsf{pk}^1_\Sigma, \dots, \mathsf{pk}^l_\Sigma)$,
        $\mathsf{pk_{Sig}} = \mathsf{pk}_\Sigma$, $\mathsf{pk_{San}} = (\mathsf{pk}_\Pi, \mathsf{pk}'_\Sigma)$
$t \leftarrow (0, m, S, \mathsf{pk_{Sig}}, \mathsf{pk_{San}})$
$d_1 \leftarrow \Sigma.\mathsf{Verify}(\mathsf{pp}_\Sigma, t, \sigma_{\mathrm{MSG}}, \mathsf{pk}_\Sigma)$
$d_2 \leftarrow \Sigma.\mathsf{Verify}(\mathsf{pp}_\Sigma, t, \sigma_{\mathrm{MSG}}, \mathsf{pk}'_\Sigma)$
$u \leftarrow (1, \mathsf{pk_{San}}, V, C)$
$d_3 \leftarrow \Sigma.\mathsf{Verify}(\mathsf{pp}_\Sigma, u, \sigma_{\mathrm{ADM}}, \mathsf{pk}_\Sigma)$
if $(d_1 = \bot \ \wedge \ d_2 = \bot) \ \vee \ d_3 = \bot \ \vee$
  $\Sigma.\mathsf{Verify}(\mathsf{pp}_\Sigma, m[i], \sigma^i, \mathsf{pk}^i_\Sigma) = \bot$
  for some $1 \le i \le l$ then
    return $\bot$
return $\top$
```

```
$\mathsf{SSS.Judge}(\mathsf{pp_{SSS}}, m, \sigma, \mathsf{pk_{Sig}}, \mathsf{pk_{San}}, \pi)$:
parse $\mathsf{pp_{SSS}} = (\mathsf{pp}_\Pi, \mathsf{pp}_\Sigma, \mathcal{M})$,
        $\sigma = (S, \sigma_{\mathrm{MSG}}, V, C, \sigma_{\mathrm{ADM}})$,
        $\mathsf{pk_{Sig}} = \mathsf{pk}_\Sigma$, $\mathsf{pk_{San}} = (\mathsf{pk}_\Pi, \mathsf{pk}'_\Sigma)$
$t \leftarrow (0, m, S, \mathsf{pk_{Sig}}, \mathsf{pk_{San}})$
$d_1 \leftarrow \Sigma.\mathsf{Verify}(\mathsf{pp}_\Sigma, t, \sigma_{\mathrm{MSG}}, \mathsf{pk}_\Sigma)$
$d_2 \leftarrow \Sigma.\mathsf{Verify}(\mathsf{pp}_\Sigma, t, \sigma_{\mathrm{MSG}}, \mathsf{pk}'_\Sigma)$
if $d_1 = \top \ \wedge \ d_2 = \bot$ then
    return $\mathsf{Sig}$
if $d_1 = \bot \ \wedge \ d_2 = \top$ then
    return $\mathsf{San}$
return $\bot$
```

```
$\mathsf{SSS.Sanit}(\mathsf{pp_{SSS}}, m, \sigma, \mathsf{sk_{San}}, \mathsf{pk_{Sig}}, \mathsf{pk_{San}}, \mathrm{MOD})$:
if $\mathrm{MOD}(m) = \bot$ then
    return $\bot$
parse $\mathsf{pp_{SSS}} = (\mathsf{pp}_\Pi, \mathsf{pp}_\Sigma, \mathcal{M})$, $m = (m[1], \dots, m[l])$,
        $\sigma = (S, \sigma_{\mathrm{MSG}}, V, C, \sigma_{\mathrm{ADM}})$, $V = (\mathsf{pk}^1_\Sigma, \dots, \mathsf{pk}^l_\Sigma)$,
        $\mathsf{sk_{San}} = (\mathsf{sk}_\Pi, \mathsf{sk}'_\Sigma)$, $\mathsf{pk_{Sig}} = \mathsf{pk}_\Sigma$,
        $\mathsf{pk_{San}} = (\mathsf{pk}_\Pi, \mathsf{pk}'_\Sigma)$, $\mathrm{MOD} = (M, l)$,
        $M = \{(i_1, m_1), \dots, (i_k, m_k)\}$
$t \leftarrow \Pi.\mathsf{Dec}(\mathsf{pp}_\Pi, C, \mathsf{pk}_\Pi, \mathsf{sk}_\Pi)$
parse $t = (\mathsf{pk}'_{\mathsf{Sig}}, K_{\mathrm{ADM}}, \mathrm{ADM})$,
        $K_{\mathrm{ADM}} = (K^1_{\mathrm{ADM}}, \dots, K^l_{\mathrm{ADM}})$,
        with $K^i_{\mathrm{ADM}} = \mathsf{sk}^i_\Sigma$ for $i \in \mathrm{ADM}$
$d_1 \leftarrow \mathsf{SSS.Verify}(\mathsf{pp_{SSS}}, m, \sigma, \mathsf{pk_{Sig}}, \mathsf{pk_{San}})$
if $d_1 = \bot \ \vee \ \mathrm{MOD}(\mathrm{ADM}) = \bot \ \vee \ \mathsf{pk}'_{\mathsf{Sig}} \neq \mathsf{pk_{Sig}}$ then
    return $\bot$
$m' \leftarrow \mathrm{MOD}(m)$
for $1 \le j \le k$ do
    $\sigma^{i_j} \leftarrow_\$ \Sigma.\mathsf{Sign}(\mathsf{pp}_\Sigma, m_j, \mathsf{sk}^{i_j}_\Sigma, \mathsf{pk}^{i_j}_\Sigma)$
$S' \leftarrow (\sigma^1, \dots, \sigma^l)$
$u \leftarrow (0, m', S', \mathsf{pk_{Sig}}, \mathsf{pk_{San}})$
$\sigma'_{\mathrm{MSG}} \leftarrow_\$ \Sigma.\mathsf{Sign}(\mathsf{pp}_\Sigma, u, \mathsf{sk}'_\Sigma, \mathsf{pk}'_\Sigma)$
$\sigma' \leftarrow (S', \sigma'_{\mathrm{MSG}}, V, C, \sigma_{\mathrm{ADM}})$
$d_2 \leftarrow \mathsf{SSS.Verify}(\mathsf{pp_{SSS}}, m', \sigma', \mathsf{pk_{Sig}}, \mathsf{pk_{San}})$
if $d_2 = \bot$ then
    return $\bot$
return $\sigma'$
```

```
$\mathsf{SSS.Proof}(\mathsf{pp_{SSS}}, m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, \mathsf{sk_{Sig}}, \mathsf{pk_{Sig}}, \mathsf{pk_{San}})$:
return $\bot$
```

Figure 5: Invisible sanitizable signature scheme from a public-key encryption scheme: verification, sanitization, judge and proof algorithms.

For sanitizing correctness, observe that the sanitizer fails to recover the plaintext $(\mathsf{pk_{Sig}}, K_{\mathrm{ADM}}, \mathrm{ADM})$ from C only with negligible probability by the correctness of Π. Given that the modification he is performing is admissible, he has all the necessary information to update the signature, and therefore the validity of the final signature σ' again follows from the correctness of Σ.

Finally, proof correctness directly follows from the correctness and the unforgeability assumptions on Σ: by these two properties, with all but negligible probability exactly one of d_1 and d_2 equals $\top$, and this allows Judge to correctly identify the party who put the signature. □

Lemma 5.4 *If the signature scheme Σ is unforgeable, then the sanitizable signature scheme* SSS *in Construction 5.1 is unforgeable.*

The idea of the proof is to note that a successful forgery against given public keys $\mathsf{pk_{Sig}}$ and $\mathsf{pk_{San}}$ requires in particular to produce a forgery against the σ_{MSG} part of the signature. In terms of concrete security we show that for any adversary $\mathcal{A}$ against unforgeability of the sanitizable scheme we can construct an adversary $\mathcal{B}$ with roughly the same running time, making at most twice as many signature requests as the number of signature and sanitization queries made by $\mathcal{A}$, and

$$\mathbf{Adv}^{\mathsf{Unf}}_{\mathcal{A},\mathsf{SSS}}(\lambda) \leq 2 \cdot \mathbf{Adv}^{\mathsf{Unf}}_{\mathcal{B},\Sigma}(\lambda).$$

Proof. Unforgeability of the sanitizable signature scheme follows from the unforgeability of the underlying signature scheme Σ. Let $\mathcal{A}$ be an adversary playing the unforgeability game for SSS. Recall that $\mathcal{A}$ can make any signature and sanitization queries, but at the end must output a forged signature σ^* of a message m^* such that it has not called the signing oracle about $(\mathsf{pk}_{\mathsf{San}}, m^*)$, nor the sanitizing oracle on $(\mathsf{pk}_{\mathsf{Sig}}, m^*)$. Since a full signature contains the signature σ_{MSG} over the public keys and the message m (in addition to the tuple of signatures S), it is straightforward to turn $\mathcal{A}$ into an adversary $\mathcal{B}$ against the unforgeability of Σ.

Adversary $\mathcal{B}$ receives a tuple of public parameters pp_Σ and a verification key pk as input, and can query a signing oracle for the corresponding secret key sk. In order to simulate the unforgeability experiment on SSS to $\mathcal{A}$, adversary $\mathcal{B}$ generates another signature key pair $(\mathsf{pk}_\Sigma, \mathsf{sk}_\Sigma) \leftarrow_\$ \Sigma.\mathsf{KGen}(\mathsf{pp}_\Sigma)$, and then runs $\mathsf{pp}_\Pi \leftarrow_\$ \Pi.\mathsf{PGen}(1^\lambda)$ to get a key pair $(\mathsf{pk}_\Pi, \mathsf{sk}_\Pi) \leftarrow_\$ \Pi.\mathsf{KGen}(\mathsf{pp}_\Pi)$ for the encryption scheme Π. He then assembles $\mathsf{pp}_{\mathsf{SSS}} = (\mathsf{pp}_\Pi, \mathsf{pp}_\Sigma, \mathcal{M}^l_\Sigma)$, and sets either $\mathsf{pk}_{\mathsf{Sig}} \leftarrow \mathsf{pk}$, $\mathsf{pk}_{\mathsf{San}} \leftarrow (\mathsf{pk}_\Pi, \mathsf{pk}_\Sigma)$ or $\mathsf{pk}_{\mathsf{Sig}} \leftarrow \mathsf{pk}_\Sigma$, $\mathsf{pk}_{\mathsf{San}} \leftarrow (\mathsf{pk}_\Pi, \mathsf{pk})$, the choice made at random. Finally, he hands $(\mathsf{pp}_{\mathsf{SSS}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$ to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks for a signature or a sanitization, $\mathcal{B}$ computes the signature as described in Figures 4 and 5, using either the secret keys sk_Σ and sk_Π he knows or calling his external oracle for a signature under sk. When $\mathcal{A}$ eventually outputs a forgery $\sigma^* = (S^*, \sigma^*_{\mathrm{MSG}}, V^*, C^*, \sigma^*_{\mathrm{ADM}})$ for a fresh message m^*, algorithm $\mathcal{B}$ extracts S^* and σ^*_{MSG} from σ^* and outputs $(0, m^*, S^*, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$ as his message, with σ^*_{MSG} being the forged signature.

For the analysis note that $\mathcal{A}$ cannot have queried the signing oracle about $(\mathsf{pk}_{\mathsf{San}}, m^*)$ nor the sanitizing oracle about $(\mathsf{pk}_{\mathsf{Sig}}, m^*)$, which means that $\mathcal{B}$ has not called its external oracle about the output message $(0, m^*, S^*, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$ either. Here we use that the messages signed for the "message" and for the "administrative" parts use different prefix bits: in case we have $\mathsf{pk}_{\mathsf{Sig}} \leftarrow \mathsf{pk}$, which means that $\mathcal{B}$ must call his external oracle twice for each signature query by $\mathcal{A}$ (once each for the "message" and for the "administrative" part), the prefix bits guarantee that these external calls cannot interfere with the output message.

We thus conclude that whenever $\mathcal{A}$ forges successfully, and if the forgery happens under the correct secret key (i.e., under sk and not sk_Σ, which happens with probability 1/2), then $\mathcal{B}$ succeeds too. ☐

Lemma 5.5 *If the signature scheme Σ is unforgeable, then the sanitizable signature scheme* SSS *in Construction 5.1 is immutable.*

In contrast to the previous case, the adversary may now impersonate the sanitizer. Hence, it may be able to produce arbitrary (sanitized) signatures for the "message" part. Still, since the signature scheme Σ is unforgeable, the adversary cannot forge the "administrative" part and can thus only rely on previously obtained signatures σ_{ADM} over given verification keys V and admissible data K_{ADM} and ADM encrypted under C. A successful attack presumes that the adversary must be able to change some inadmissible part of the message, for which K_{ADM} does not contain the appropriate key material. This requires the adversary to forge a signature for some public key in V.

In terms of concrete security we get that for any adversary $\mathcal{A}$ against immutability of the sanitizable scheme, making at most q signature queries of messages consisting of l blocks each, we have an adversary $\mathcal{B}$ with roughly the same running time, making at most twice as many signature requests as $\mathcal{A}$, and an adversary $\mathcal{C}$ such that

$$\mathbf{Adv}^{\mathsf{Imm}}_{\mathcal{A},\mathsf{SSS}}(\lambda) \leq \mathbf{Adv}^{\mathsf{Unf}}_{\mathcal{B},\Sigma}(\lambda) + lq \cdot \mathbf{Adv}^{\mathsf{Unf}}_{\mathcal{C},\Sigma}(\lambda).$$

Proof. Recall that, in the immutability experiment, an adversary $\mathcal{A}$ is given a tuple $\mathsf{pp}_{\mathsf{SSS}}$ of public parameters for a sanitizable signature scheme and a signer public key $\mathsf{pk}_{\mathsf{Sig}}$. It is allowed to submit queries of the form $(m, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM})$ to a signing oracle, and its goal is to create a signature σ^* for a message m^* and sanitizer key $\mathsf{pk}^*_{\mathsf{San}}$ such that $(\mathsf{pk}^*_{\mathsf{San}}, m^*)$ is not among all admissible modifications $(\mathsf{pk}_{\mathsf{San}}, \mathrm{MOD}(m))$ with $\mathrm{MOD}(\mathrm{ADM}) = \top$ of the queries to the signing oracle.

In this proof we assume that $\mathcal{A}$ makes at most q signature queries in total, with each message having l blocks. We first claim that, if $\mathcal{A}$ successfully produces a signature $\sigma^* = (S^*, \sigma^*_{\mathrm{MSG}}, V^*, C^*, \sigma^*_{\mathrm{ADM}})$ as above, then σ^*_{ADM} must be the signature component of an "administrative" part of a signature previously obtained from the signing oracle. In other words, σ^*_{ADM} cannot have been generated by $\mathcal{A}$ itself, but must be the signature of a message of the form $(1, \mathsf{pk}_{\mathsf{San}}, V^*, C^*)$ previously signed under $\mathsf{sk}_{\mathsf{Sig}}$ by the signing oracle.

Indeed, if this were not the case, we would get a reduction $\mathcal{B}$ to the unforgeability of Σ: adversary $\mathcal{B}$ receives a tuple of public parameters pp_{Σ} and a verification key pk as input, and can query a signing oracle for the corresponding secret key sk. In order to simulate the immutability experiment on SSS to $\mathcal{A}$, adversary $\mathcal{B}$ runs $\mathsf{pp}_{\Pi} \leftarrow_{\$} \Pi.\mathsf{PGen}(1^{\lambda})$, assembles $\mathsf{pp}_{\mathsf{SSS}} = (\mathsf{pp}_{\Pi}, \mathsf{pp}_{\Sigma}, \mathcal{M}^l_{\Sigma})$, and sets $\mathsf{pk}_{\mathsf{Sig}} \leftarrow \mathsf{pk}$. Finally, he hands $(\mathsf{pp}_{\mathsf{SSS}}, \mathsf{pk}_{\mathsf{Sig}})$ to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks for a signature, $\mathcal{B}$ internally performs all the steps described in Figure 4, except for the creation fo σ_{MSG} and σ_{ADM}, where he calls his external signing oracle for the secret key sk.

When $\mathcal{A}$ eventually outputs a signature $\sigma^* = (S^*, \sigma^*_{\mathrm{MSG}}, V^*, C^*, \sigma^*_{\mathrm{ADM}})$ for a message m^* and sanitizer public key $\mathsf{pk}^*_{\mathsf{San}}$ as above, algorithm $\mathcal{B}$ extracts V^*, C^* and σ^*_{ADM} from σ^* and outputs $(1, \mathsf{pk}^*_{\mathsf{San}}, V^*, C^*)$ as his message, with σ^*_{ADM} being the forged signature.

For the analysis of $\mathcal{B}$'s success probability, first note that the simulation is perfect from $\mathcal{A}$'s point of view. Furthermore, the tuple $(1, \mathsf{pk}^*_{\mathsf{San}}, V^*, C^*)$ has not been previously signed by the simulated signer for any signature σ_{ADM}, because of our initial assumption. Since we use a distinct prefix bit '0' for the signatures σ_{MSG}, algorithm $\mathcal{B}$ cannot have called its external oracle about the tuple when generating a signature part σ_{MSG} either. It follows that a valid forgery for such a fresh "administrative" part gives rise to a successful attack on the underlying signature scheme Σ.

In conclusion, $\mathcal{A}$ must reuse the data $(\mathsf{pk}^*_{\mathsf{San}}, V^*, C^*) = (\mathsf{pk}_{\mathsf{San}}, V, C)$ from some previous signature query for a message m and admissible operations ADM. Now, if ADM allowed to alter the entire message m, that is, $\mathrm{ADM} = (\{1, 2, \ldots, l\}, l)$, then it would be impossible for $\mathcal{A}$ to find an inadmissible modification to break immutability. Hence, there must be some index $1 \leq j \leq l$ with $j \notin \mathrm{ADM}$, but such that $\mathcal{A}$ still is able to modify $m[j]$.

We now show a reduction $\mathcal{C}$ to the unforgeability of Σ. Again, adversary $\mathcal{C}$ receives a tuple of public parameters pp_{Σ} and a verification key pk as input, and can query a signing oracle for the corresponding secret key sk. At the outset of the simulation of the immutability experiment to $\mathcal{A}$, adversary $\mathcal{C}$ guesses the query h (among the q signature queries) adversary $\mathcal{A}$ will later reuse to extract the "administrative" part of the signature, and also the position j (among the l possible blocks) that will be inadmissible, but that $\mathcal{A}$ still manages to modify. $\mathcal{C}$ then prepares the simulation of the immutability experiment: He runs $\mathsf{pp}_{\Pi} \leftarrow_{\$} \Pi.\mathsf{PGen}(1^{\lambda})$, assembles $\mathsf{pp}_{\mathsf{SSS}} = (\mathsf{pp}_{\Pi}, \mathsf{pp}_{\Sigma}, \mathcal{M}^l_{\Sigma})$, and sets $\mathsf{pk}_{\mathsf{Sig}} \leftarrow \mathsf{pk}_{\Sigma}$ for a self-generated key pair $(\mathsf{pk}_{\Sigma}, \mathsf{sk}_{\Sigma}) \leftarrow_{\$} \Sigma.\mathsf{KGen}(\mathsf{pp}_{\Sigma})$. Finally, he hands $(\mathsf{pp}_{\mathsf{SSS}}, \mathsf{pk}_{\mathsf{Sig}})$ to $\mathcal{A}$.

Whenever $\mathcal{A}$ queries his signature oracle, $\mathcal{C}$ generates all the values by himself as per Figure 4—except for query number h, where the verification key he uses for the j-th block is pk (all the other keys still being generated by $\mathcal{C}$). Algorithm $\mathcal{C}$ aborts the simulation if, on query h, adversary $\mathcal{A}$ submits a parameter ADM with $j \in \mathrm{ADM}$, because in that case $\mathcal{C}$ is unable to generate the tuple K_{ADM} (since he doesn't know the secret key corresponding to pk). On the other hand, if $j \notin \mathrm{ADM}$, the secret key sk corresponding to pk is not included in K_{ADM}, and therefore the simulation can be carried out correctly. If $\mathcal{A}$ eventually outputs a signature $\sigma^* = (S^*, \sigma^*_{\mathrm{MSG}}, V^*, C^*, \sigma^*_{\mathrm{ADM}})$ for a message m^*, then $\mathcal{C}$ parses $S^* = (\sigma^1, \ldots, \sigma^l)$ and then outputs $m^*[j]$ as his message, with σ^j being the forged signature.

For the analysis note that, as discussed above, adversary $\mathcal{A}$ must reuse the value V created in some previous signature query, and the parameter ADM in that query must have at least one index $j \notin$ ADM. Then observe that, since $\mathcal{C}$ has made no oracle queries at all, $\mathcal{C}$ wins the unforgeability experiment exactly if the data reused by $\mathcal{A}$ are those from the h-th query, and if block number j was not admissible in that query, but still $m^*[j]$ is different from the originally queried message block. Indeed, for a signature to be valid, all signatures in S must verify under the corresponding public keys, and therefore $\mathcal{C}$ has indeed successfully forged a signature. We thus conclude that $\mathcal{C}$'s advantage equals $\mathcal{A}$'s advantage, times the probability of $\mathcal{C}$ correctly guessing the values of h and j, which is $1/lq$. □

Lemma 5.6 *The sanitizable signature scheme* SSS *in Construction 5.1 is perfectly private.*

Privacy says that one cannot distinguish the process of signing and then sanitizing any of two messages if they result in the same final message. For our scheme this follows as this process creates a distribution which only depends on the final message.

Proof. Recall that privacy means that the adversary can send a query of the form $(m_0, \mathrm{MOD}_0, m_1, \mathrm{MOD}_1,$ ADM) to a left-or-right oracle, provided that both modifications are admissible and that the modified messages coincide, i.e. $\mathrm{MOD}_0(m_0) = \mathrm{MOD}_1(m_1)$. Based on a secret bit $b \in \{0,1\}$, the oracle then first signs the message m_b and then sanitizes the message and signature to obtain a sanitized signature for $\mathrm{MOD}(m_b)$. The adversary's task is to predict b, even when given access to signing and a sanitizing oracles.

To see that our scheme is perfectly private, recall that signatures in SSS consist of two parts. The "administrative" part $(V, C, \sigma_{\mathrm{ADM}})$ only depends on the ephemeral keys V and K_{ADM}, as well as $\mathsf{pk}_{\mathsf{Sig}}$, $\mathsf{sk}_{\mathsf{Sig}}$, $\mathsf{pk}_{\mathsf{San}}$, ADM, and l, and is thus independent of the message except for its length. The "message" part $(S, \sigma_{\mathrm{MSG}})$ only depends on the ephemeral keys used in the signing process, $\mathsf{sk}_{\mathsf{Sig}}$ or $\mathsf{sk}_{\mathsf{San}}$, $\mathsf{pk}_{\mathsf{Sig}}$, $\mathsf{pk}_{\mathsf{San}}$, and the message itself, but not on the potential history of the message or the sanitization process. Hence, for any left-or-right query yielding the same message, the sanitized signature has the same distribution for both values of $b \in \{0,1\}$. Furthermore, each signature uses fresh random coins, such that the additional oracles for signing and sanitization cannot help to distinguish the two cases either. □

Lemma 5.7 *If the signature scheme* Σ *is unforgeable then the sanitizable signature scheme* SSS *in Construction 5.1 is publicly accountable.*

Non-interactive public ccountability is straightforward, as it requires an adversary $\mathcal{A}$ to impersonate the legitimate signer or sanitizer, and to forge a signature under the other party's public key. Every such adversary $\mathcal{A}$ can be turned into an adversary $\mathcal{B}$ against the unforgeability property of Σ, with approximately the same running time and making at most twice as many signature queries, and such that

$$\mathbf{Adv}^{\mathsf{NIPA}}_{\mathcal{A},\mathsf{SSS}}(\lambda) \leq 2 \cdot \mathbf{Adv}^{\mathsf{Unf}}_{\mathcal{B},\Sigma}(\lambda).$$

Proof. Recall that the Judge algorithm checks if the signature part σ_{MSG} verifies under the signer or sanitizer public key, and points towards the corresponding party. The adversary, trying to impersonate either the signer or the sanitizer, tries to make the judge point to the other party for some m^* and σ^*, while trivial wins are of course excluded. Since the signature σ^*_{MSG} is over both public keys and the message m^*, the adversary can only succeed if it forges a signature under the corresponding public key of the other party.

More precisely, let $\mathcal{A}$ be an adversary playing the non-interactive public accountability game for SSS. We construct an adversary $\mathcal{B}$ against the unforgeability of Σ as follows: $\mathcal{B}$ receives as input a tuple of public parameters pp_Σ and a verification key pk, and can query a signing oracle for the corresponding secret key sk. He then generates another signature key pair $(\mathsf{pk}_\Sigma, \mathsf{sk}_\Sigma)$, and creates public parameters pp_Π

for Π to get a key pair $(\mathsf{pk}_\Pi, \mathsf{sk}_\Pi)$ for the encryption scheme. Finally, he assembles $\mathsf{pp}_{\mathsf{SSS}}$ as usual, sets either $\mathsf{pk}_{\mathsf{Sig}} \leftarrow \mathsf{pk}$, $\mathsf{pk}_{\mathsf{San}} \leftarrow (\mathsf{pk}_\Pi, \mathsf{pk}_\Sigma)$ or $\mathsf{pk}_{\mathsf{Sig}} \leftarrow \mathsf{pk}_\Sigma$, $\mathsf{pk}_{\mathsf{San}} \leftarrow (\mathsf{pk}_\Pi, \mathsf{pk})$ (the choice made at random), and hands $(\mathsf{pp}_{\mathsf{SSS}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$ to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks for a signature or a sanitization, $\mathcal{B}$ computes the signature as described in Figures 4 and 5, using either the secret keys sk_Σ and sk_Π he knows or calling his external oracle for a signature under sk. When $\mathcal{A}$ eventually outputs a triple $(m^*, \sigma^*, \mathsf{pk}^*)$, algorithm $\mathcal{B}$ extracts S^* and σ^*_{MSG} from σ^* and outputs $(0, m^*, S^*, \mathsf{pk}^*, \mathsf{pk}_{\mathsf{San}})$ or $(0, m^*, S^*, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}^*)$ (depending on whether $\mathcal{A}$ impersonated the signer or the sanitizer), with σ^*_{MSG} being the forged signature.

For the analysis, first note that $\mathcal{B}$ can easily verify if $\mathcal{A}$ wins the public accountability experiment, given that the Verify and Judge algorithms can both be run publicly, and that he knows which queries he has answered. Furthermore, the condition that $\mathcal{A}$'s final output cannot be the result of an oracle query guarantees that $\mathcal{B}$ has not queried his output message to his oracle before. Therefore, $\mathcal{B}$ indeed outputs a forgery, provided that this is indeed a signature under sk (and not sk_Σ), i.e. that he has correctly guessed at the beginning of the experiment which of the two parties $\mathcal{A}$ has impersonated. We thus conclude that, whenever $\mathcal{A}$ successfully breaks public accountability, and if he impersonates the correct party (which happens with probability 1/2), then $\mathcal{B}$ succeeds too. □

Lemma 5.8 *If the encryption scheme Π is $\mathsf{IND{-}CPA}$-secure, then the sanitizable signature scheme SSS in Construction 5.1 is invisible.*

Invisibility follows from the security of the encryption scheme, since distinguishing the signatures for ADM_0 and ADM_1 corresponds to distinguish the corresponding encryptions. In terms of concrete security, any adversary $\mathcal{A}$ against invisibility, making at most q left-or-right queries, can be transformed into an $\mathsf{IND{-}CPA}$-adversary $\mathcal{B}$ with almost the same running time as $\mathcal{A}$ and

$$\mathbf{Adv}^{\mathsf{Inv}}_{\mathcal{A},\mathsf{SSS}}(\lambda) \le q \cdot \mathbf{Adv}^{\mathsf{IND-CPA}}_{\mathcal{B},\Pi}(\lambda).$$

Here, the factor q loss stems from our definition of $\mathsf{IND{-}CPA}$-security for a *single* challenge ciphertext, whereas in the proof below we use security against multiple challenge queries. This is known to follow from the single-ciphertext case via a hybrid argument, losing a factor q in the advantage.

Proof. Invisibility states that an adversary $\mathcal{A}$ can query a left-or-right oracle which, based on a secret bit $b \in \{0,1\}$, signs a message m making one of two sets of sanitizing rights, ADM_0 or ADM_1, admissible in the final signature σ. The resulting signature gets whitelisted together with the message m and the intersection $\mathrm{ADM}_0 \cap \mathrm{ADM}_1$. The adversary can also query a sanitizing oracle for any whitelisted data $(m, \sigma, \mathrm{ADM})$ and matching modification to get a sanitized signature σ' for m', and $(m', \sigma', \mathrm{ADM})$ gets whitelisted as well. $\mathcal{A}$ wins the experiment if he correctly guesses b.

We argue that a successful distinguisher $\mathcal{A}$ against invisibility yields a successful attacker $\mathcal{B}$ against $\mathsf{IND{-}CPA}$-security of Π. This is done by a reduction. Algorithm $\mathcal{B}$ receives as input a tuple of public parameters pp_Π and a public key pk of the encryption scheme Π, and starts simulating the invisibility experiment to $\mathcal{A}$: he generates public parameters $\mathsf{pp}_\Sigma \leftarrow_\$ \Sigma.\mathsf{PGen}(1^\lambda)$ and two pairs of signature keys $(\mathsf{pk}_\Sigma, \mathsf{sk}_\Sigma), (\mathsf{pk}'_\Sigma, \mathsf{sk}'_\Sigma) \leftarrow_\$ \Sigma.\mathsf{KGen}(\mathsf{pp}_\Sigma)$, assembles $\mathsf{pp}_{\mathsf{SSS}} = (\mathsf{pp}_\Pi, \mathsf{pp}_\Sigma, \mathcal{M}^l_\Sigma)$, then sets $\mathsf{pk}_{\mathsf{Sig}} \leftarrow \mathsf{pk}_\Sigma$ and $\mathsf{pk}_{\mathsf{San}} \leftarrow (\mathsf{pk}, \mathsf{pk}'_\Sigma)$, and finally hands $(\mathsf{pp}_{\mathsf{SSS}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$ to $\mathcal{A}$.

Every time adversary $\mathcal{A}$ submits a query $(m, \mathrm{ADM}_0, \mathrm{ADM}_1)$ to his left-or-right oracle, algorithm $\mathcal{B}$ emulates the genuine left-or-right oracle and runs the same steps as the Sign algorithm, but hands $(\mathsf{pk}_{\mathsf{Sig}}, K_{\mathrm{ADM}_0}, \mathrm{ADM}_0)$ and $(\mathsf{pk}_{\mathsf{Sig}}, K_{\mathrm{ADM}_1}, \mathrm{ADM}_1)$ to his external left-or-right encryption oracle and uses the returned ciphertext C to assemble the signature.

For every sanitization request $(m, \sigma, \mathrm{MOD})$, algorithm $\mathcal{B}$ can check if the whitelist contains an entry $(m, \sigma, \mathrm{ADM})$, and in case run the sanitization algorithm using the secret keys sk^i_Σ corresponding to the verification keys contained in V ($\mathcal{B}$ knows these secret keys because he has generated them when simulating the left-or-right oracle). When $\mathcal{A}$ finally outputs a guess for b, adversary $\mathcal{B}$ simply returns the same bit.

For the analysis note that $\mathcal{B}$'s calls to its left-or-right oracle always contain equal-length messages $(\mathsf{pk}_{\mathsf{Sig}}, K_{\mathrm{ADM}_i}, \mathrm{ADM}_i)$, since we assume that all keys of the scheme Σ are of fixed length L and that the sanitizing rights are encoded in a length-invariant manner. It follows that the respective advantages of the two adversaries coincide. $\square$

5.3 Achieving Strong Invisibility

In the previous sections we have shown that invisibility is equivalent to $\mathsf{IND{-}CPA}$-secure encryption, and that strong invisibility implies $\mathsf{IND{-}CCA2}$-secure encryption. Here we show that the latter implication also holds in the other direction: If we use an $\mathsf{IND{-}CCA2}$-secure encryption scheme in our construction, then we get a strongly invisible sanitizable signature scheme.

Theorem 5.9 *If the signature scheme Σ is correct and unforgeable, and the encryption scheme Π is correct, then the sanitizable signature scheme SSS in Construction 5.1 is correct. If Σ is unforgeable and Π is $\mathsf{IND{-}CCA2}$-secure, then SSS is unforgeable, immutable, private, publicly accountable, and strongly invisible.*

Except for invisibility, all other security properties hold as before. Hence, it is sufficient to prove strong invisibility. The concrete bounds are identical to the ones in the $\mathsf{IND{-}CPA}$ case: The advantage of an adversary against strong invisibility is bounded by (q times) the advantage against $\mathsf{IND{-}CCA2}$-security of the underlying encryption scheme.

Proof. The difference between ordinary and strong invisibility is that the adversary $\mathcal{A}$ can now also ask the left-or-right oracle about sanitizer keys $\mathsf{pk}'_{\mathsf{San}} \neq \mathsf{pk}_{\mathsf{San}}$ (in which case he must set $\mathrm{ADM}_0 = \mathrm{ADM}_1$), and the sanitization oracle about signer keys $\mathsf{pk}'_{\mathsf{Sig}} \neq \mathsf{pk}_{\mathsf{Sig}}$, where $\mathsf{pk}_{\mathsf{Sig}}$ and $\mathsf{pk}_{\mathsf{San}}$ are the given keys. When doing the former, the output of such queries is not whitelisted, and in the latter case the adversary always receives a sanitized signature. We argue that we can lift the reduction $\mathcal{B}$ from the $\mathsf{IND{-}CPA}$ case to the $\mathsf{IND{-}CCA2}$ case and cover the extended capabilities available to $\mathcal{A}$ through strong invisibility.

We start by observing that if $\mathcal{A}$ queries the left-or-right oracle with $\mathsf{pk}'_{\mathsf{San}} \neq \mathsf{pk}_{\mathsf{San}}$, then the condition $\mathrm{ADM}_0 = \mathrm{ADM}_1$ means that $\mathcal{B}$ doesn't actually need to know the bit b to answer the query: he just runs the Sign algorithm as defined in Figure 4 (with $\mathrm{ADM} = \mathrm{ADM}_0 = \mathrm{ADM}_1$) and returns the signature to $\mathcal{A}$. This already covers the new signing queries.

For the new sanitization queries (i.e., with $\mathsf{pk}'_{\mathsf{Sig}} \neq \mathsf{pk}_{\mathsf{Sig}}$), some care must be taken. In fact, given that $\mathcal{A}$ now knows the secret key $\mathsf{sk}'_{\mathsf{Sig}}$, he could first ask his left-or-right oracle for a genuine signature $\sigma = (S, \sigma_{\mathrm{MSG}}, V, C, \sigma_{\mathrm{ADM}})$ (with $\mathsf{pk}'_{\mathsf{San}} = \mathsf{pk}_{\mathsf{San}}$), then extract the ciphertext C and embed it into a newly forged signature (under $\mathsf{sk}'_{\mathsf{Sig}}$). Afterwards, when submitting this signature to the sanitization oracle, he could learn the bit b simply observing which blocks have been modified. To overcome this problem, we have included the signer key $\mathsf{pk}_{\mathsf{Sig}}$ in the encryption C.

$\mathcal{B}$ again mimics the genuine sanitization oracle and executes the Sanit algorithm as detailed in Figure 5. We distinguish two cases. If such a query carries a ciphertext C created by the genuine signer (or, for that matter, contains a public key different from $\mathsf{pk}'_{\mathsf{Sig}}$) then, because the original signer public key is encrypted in C and is verified against the given $\mathsf{pk}'_{\mathsf{Sig}}$ (see the definition of Sanit in Figure 5), the genuine sanitization oracle would return $\perp$. Since $\mathcal{B}$ can easily verify this (either by calling his decryption oracle, or because he remembers the cipertexts he has issued), it can answer accordingly. If on the other hand C contains the public key $\mathsf{pk}'_{\mathsf{Sig}} \neq \mathsf{pk}_{\mathsf{Sig}}$, the switch to an $\mathsf{IND{-}CCA2}$-game allows $\mathcal{B}$ to execute Sanit as defined in Figure 5 (with the call to $\Pi.\mathsf{Dec}$ substituted with his decryption oracle), and compute the answer accordingly.

Overall, $\mathcal{B}$ again runs a perfect simulation and $\mathcal{A}$'s advantage in breaking strong invisibility is bounded from above by $\mathcal{B}$'s advantage against the $\mathsf{IND{-}CCA2}$-security of the encryption scheme. $\square$

6 Conclusions

Our results show that building invisible sanitizable signature schemes from one-way functions alone is presumably hard, since deriving public-key encryption from one-wayness in a black-box way is infeasible [IR89]. This is in contrast to sanitizable schemes without the extra property of invisibility. Namely, Brzuska *et al.* [BFLS09] gave a simple construction of a "non-invisible" scheme based on regular signature schemes only.

An interesting open question concerns the minimal assumptions required to achieve *transparency* for sanitizable signatures, independently of the question regarding invisibility. Recall that transparency can be somewhat thought of as the opposite of accountability, trying to hide whether the signer or the sanitizer has created the signatures. Achieving accountability and the other common security properties, except for transparency (and except for invisibility, of course), is possible using one-way functions alone [BFLS09, BPS13]. Current constructions achieving transparency are based on assumptions seemingly stronger than one-way functions, such a group signature schemes [BFLS10], zero-knowledge proofs [FKM$^+$16], or (chameleon) hash functions [BCD$^+$17, CDK$^+$17]. Finally, for a sanitizable signature scheme to be both transparent and invisble, public-key encryption is at least necessary as discussed here.

Acknowledgments

We thank the anonymous reviewers for their valuable comments and suggestions. This work has been co-funded by the DFG as part of project P2 within the CRC 1119 CROSSING.

References

[ABC$^+$12] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, abhi shelat, and Brent Waters. Computing on authenticated data. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 1–20, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany. (Cited on page 5.)

[ACdT05] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS 2005: 10th European Symposium on Research in Computer Security*, volume 3679 of *Lecture Notes in Computer Science*, pages 159–177, Milan, Italy, September 12–14, 2005. Springer, Heidelberg, Germany. (Cited on pages 3, 5, 10, 16, 35, 37, and 38.)

[ADR02] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany. (Cited on page 33.)

[BBD$^+$10] Christina Brzuska, Heike Busch, Özgür Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, and Dominique Schröder. Redactable signatures for tree-structured data: Definitions and constructions. In Jianying Zhou and Moti Yung, editors, *ACNS 10: 8th International Conference on Applied Cryptography and Network Security*, volume 6123 of *Lecture Notes in Computer Science*, pages 87–104, Beijing, China, June 22–25, 2010. Springer, Heidelberg, Germany. (Cited on page 5.)

[BCD$^+$17] Michael Till Beck, Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Practical strongly invisible and strongly accountable sanitizable signatures. In Josef Pieprzyk and Suriadi Suriadi, editors, *ACISP 2017*, pages 437–452, 2017. (Cited on pages 3, 4, 5, 9, 10, 25, 37, and 38.)

[BFF$^+$09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 317–336, Irvine, CA, USA, March 18–20, 2009. Springer, Heidelberg, Germany. (Cited on pages 5, 6, 7, 8, 9, 10, 16, 35, 37, 38, and 44.)

[BFLS09] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Santizable signatures: How to partially delegate control for authenticated data. In *BIOSIG 2009*, volume 155 of *LNI*, pages 117–128. GI, 2009. (Cited on pages 5, 6, and 25.)

[BFLS10] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 444–461, Paris, France, May 26–28, 2010. Springer, Heidelberg, Germany. (Cited on pages 3, 5, 9, 25, and 38.)

[BPS12] Christina Brzuska, Henrich C. Pöhls, and Kai Samelin. Non-interactive public accountability for sanitizable signatures. In *EuroPKI 2012*, volume 7868 of *Lecture Notes in Computer Science (LNCS)*, pages 178–193. Springer-Verlag, 2012. (Cited on pages 5, 9, 38, and 44.)

[BPS13] Christina Brzuska, Henrich Christopher Pöhls, and Kai Samelin. Efficient and perfectly unlinkable sanitizable signatures without group signatures. In *EuroPKI 2013*, volume 8341 of *Lecture Notes in Computer Science*, pages 12–30. Springer, 2013. (Cited on pages 5, 25, and 44.)

[CDK$^+$17] Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures. In Serge Fehr, editor, *PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 10175 of *Lecture Notes in Computer Science*, pages 152–182, Amsterdam, The Netherlands, March 28–31, 2017. Springer, Heidelberg, Germany. (Cited on pages 3, 5, 6, 9, 10, 25, 37, and 38.)

[CJ10] Sébastien Canard and Amandine Jambert. On extended sanitizable signature schemes. In Josef Pieprzyk, editor, *Topics in Cryptology – CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 179–194, San Francisco, CA, USA, March 1–5, 2010. Springer, Heidelberg, Germany. (Cited on page 5.)

[CJL12] Sébastien Canard, Amandine Jambert, and Roch Lescuyer. Sanitizable signatures with several signers and sanitizers. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *AFRICACRYPT 12: 5th International Conference on Cryptology in Africa*, volume 7374 of *Lecture Notes in Computer Science*, pages 35–52, Ifrance, Morocco, July 10–12, 2012. Springer, Heidelberg, Germany. (Cited on page 5.)

[CLM08] Sébastien Canard, Fabien Laguillaumie, and Michel Milhau. Trapdoor sanitizable signatures and their application to content protection. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS 08: 6th International Conference on Applied*

Cryptography and Network Security, volume 5037 of *Lecture Notes in Computer Science*, pages 258–276, New York, NY, USA, June 3–6, 2008. Springer, Heidelberg, Germany. (Cited on page 5.)

[CMTV15] Sandro Coretti, Ueli Maurer, Björn Tackmann, and Daniele Venturi. From single-bit to multi-bit public-key encryption via non-malleable codes. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 532–560, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany. (Cited on page 4.)

[DDH$^+$15] Denise Demirel, David Derler, Christian Hanser, Henrich C. Pöhls, Daniel Slamanig, and Giulia Traverso. Overview of functional and malleable signature schemes (prismacloud deliverable d4.4). Technical report, 2015. (Cited on page 5.)

[DHO16] Ivan Damgård, Helene Haagh, and Claudio Orlandi. Access control encryption: Enforcing information flow with cryptography. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 547–576, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany. (Cited on page 5.)

[dMPPS14] Hermann de Meer, Henrich C. Pöhls, Joachim Posegga, and Kai Samelin. On the relation between redactable and sanitizable signature schemes. In Jan Jürjens, Frank Piessens, and Nataliia Bielova, editors, *ESSoS 2014*, pages 113–130, 2014. (Cited on pages 5, 37, and 38.)

[DPSS16] David Derler, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. A general framework for redactable signatures and new constructions. In Soonhak Kwon and Aaram Yun, editors, *ICISC 15: 18th International Conference on Information Security and Cryptology*, volume 9558 of *Lecture Notes in Computer Science*, pages 3–19, Seoul, Korea, November 25–27, 2016. Springer, Heidelberg, Germany. (Cited on page 5.)

[DS15] David Derler and Daniel Slamanig. Rethinking privacy for extended sanitizable signatures and a black-box construction of strongly private schemes. In Man Ho Au and Atsuko Miyaji, editors, *ProvSec 2015: 9th International Conference on Provable Security*, volume 9451 of *Lecture Notes in Computer Science*, pages 455–474, Kanazawa, Japan, November 24–26, 2015. Springer, Heidelberg, Germany. (Cited on page 5.)

[FF15] Victoria Fehr and Marc Fischlin. Sanitizable signcryption: Sanitization over encrypted data (full version). Cryptology ePrint Archive, Report 2015/765, 2015. `http://eprint.iacr.org/2015/765`. (Cited on page 5.)

[FKM$^+$16] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 9614 of *Lecture Notes in Computer Science*, pages 301–330, Taipei, Taiwan, March 6–9, 2016. Springer, Heidelberg, Germany. (Cited on page 25.)

[GGOT15] Esha Ghosh, Michael T. Goodrich, Olga Ohrimenko, and Roberto Tamassia. Fully-dynamic verifiable zero-knowledge order queries for network data. Cryptology ePrint Archive, Report 2015/283, 2015. `http://eprint.iacr.org/2015/283`. (Cited on page 5.)

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th Annual ACM Symposium on Theory of Computing*, pages 365–377, San Francisco, CA, USA, May 5–7, 1982. ACM Press. (Cited on page 31.)

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. (Cited on page 31.)

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988. (Cited on pages 33 and 37.)

[GOT15] Esha Ghosh, Olga Ohrimenko, and Roberto Tamassia. Zero-knowledge authenticated order queries and order statistics on a list. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15: 13th International Conference on Applied Cryptography and Network Security*, volume 9092 of *Lecture Notes in Computer Science*, pages 149–171, New York, NY, USA, June 2–5, 2015. Springer, Heidelberg, Germany. (Cited on page 5.)

[GQZ11] Junqing Gong, Haifeng Qian, and Yuan Zhou. Fully-secure and practical sanitizable signatures. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *INSCRYPT 2010*, pages 300–317, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cited on pages 3, 5, 9, 37, and 39.)

[HLW12] Susan Hohenberger, Allison B. Lewko, and Brent Waters. Detecting dangerous queries: A new approach for chosen ciphertext security. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 663–681, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany. (Cited on page 4.)

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, WA, USA, May 15–17, 1989. ACM Press. (Cited on pages 6 and 25.)

[JMSW02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 244–262, San Jose, CA, USA, February 18–22, 2002. Springer, Heidelberg, Germany. (Cited on page 5.)

[KL06] Marek Klonowski and Anna Lauks. Extended sanitizable signatures. In Min Surp Rhee and Byoungcheon Lee, editors, *ICISC 06: 9th International Conference on Information Security and Cryptology*, volume 4296 of *Lecture Notes in Computer Science*, pages 343–355, Busan, Korea, November 30 – December 1, 2006. Springer, Heidelberg, Germany. (Cited on page 5.)

[KSS16] Stephan Krenn, Kai Samelin, and Dieter Sommer. Stronger security for sanitizable signatures. In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, Alessandro Aldini, Fabio Martinelli, and Neeraj Suri, editors, *QASA 2015*, pages 100–117, Cham, 2016. Springer International Publishing. (Cited on pages 5, 9, 37, and 38.)

[MH15] Takahiro Matsuda and Goichiro Hanaoka. An asymptotically optimal method for converting bit encryption to multi-bit encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 415–442, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany. (Cited on page 4.)

[Ms09] Steven Myers and abhi shelat. Bit encryption is complete. In *50th Annual Symposium on Foundations of Computer Science*, pages 607–616, Atlanta, GA, USA, October 25–27, 2009. IEEE Computer Society Press. (Cited on page 4.)

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *21st Annual ACM Symposium on Theory of Computing*, pages 33–43, Seattle, WA, USA, May 15–17, 1989. ACM Press. (Cited on pages 4 and 17.)

[PSP11] Henrich Christopher Pöhls, Kai Samelin, and Joachim Posegga. Sanitizable signatures in XML signature - performance, mixing properties, and revisiting the property of transparency. In Javier Lopez and Gene Tsudik, editors, *ACNS 11: 9th International Conference on Applied Cryptography and Network Security*, volume 6715 of *Lecture Notes in Computer Science*, pages 166–182, Nerja, Spain, June 7–10, 2011. Springer, Heidelberg, Germany. (Cited on page 5.)

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd Annual ACM Symposium on Theory of Computing*, pages 387–394, Baltimore, MD, USA, May 14–16, 1990. ACM Press. (Cited on pages 4 and 17.)

[YSL10] Dae Hyun Yum, Jae Woo Seo, and Pil Joong Lee. Trapdoor sanitizable signatures made easy. In Jianying Zhou and Moti Yung, editors, *ACNS 10: 8th International Conference on Applied Cryptography and Network Security*, volume 6123 of *Lecture Notes in Computer Science*, pages 53–68, Beijing, China, June 22–25, 2010. Springer, Heidelberg, Germany. (Cited on page 5.)

A Standard Cryptographic Building Blocks

For completeness and for better clarity, we briefly review in this Appendix the standard cryptographic building blocks we have used in the main body of our work: public-key encryptions schemes, digital signature schemes, and one-way functions.

A.1 Public-Key Encryption Schemes

Definition A.1 (Public-Key Encryption Scheme) *A* public-key encryption scheme Π *is a tuple of four probabilistic polynomial-time algorithms* $\Pi := (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *defined as follows:*

$\mathsf{pp} \leftarrow_{\$} \mathsf{PGen}(1^{\lambda})$: *On input a security parameter $\lambda \in \mathbb{N}$, the algorithm* PGen *generates a tuple of public parameters of* Π. *The tuple* pp *includes the security parameter 1^{λ}, the definition of the message space $\mathcal{M}$, the ciphertext space $\mathcal{C}$, and the key space $\mathcal{K}$ (together with a special symbol $\perp \notin \mathcal{M} \cup \mathcal{C} \cup \mathcal{K}$), along with any other information needed to encrypt or decrypt messages, except for identities and user key pairs.*

$(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KGen}(\mathsf{pp})$: *On input a tuple of public parameters* pp*, the algorithm* KGen *returns a key pair or an error message, that is* $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K} \cup \{\perp\}$. *Here,* pk *and* sk *are the public encryption and the secret decryption keys, respectively. We write $\mathcal{K}_{\mathsf{pk}}$ and $\mathcal{K}_{\mathsf{sk}}$ for the sets of all possible public encryption and secret decryption keys.*

$c \leftarrow_{\$} \mathsf{Enc}(\mathsf{pp}, m, \mathsf{pk})$: *On input a tuple of public parameters* pp*, a message $m \in \mathcal{M}$, and a public encryption key $\mathsf{pk} \in \mathcal{K}_{\mathsf{pk}}$, the algorithm* Enc *returns a ciphertext or an error message, that is $c \in \mathcal{C} \cup \{\perp\}$.*

$m \leftarrow \mathsf{Dec}(\mathsf{pp}, c, \mathsf{pk}, \mathsf{sk})$: *On input a tuple of public parameters* pp*, a ciphertext $c \in \mathcal{C}$, a public encryption key $\mathsf{pk} \in \mathcal{K}_{\mathsf{pk}}$, and a secret decryption key $\mathsf{sk} \in \mathcal{K}_{\mathsf{sk}}$, the deterministic algorithm* Dec *returns a plaintext or an error message, that is $m \in \mathcal{M} \cup \{\perp\}$.*

Remark. Observe that Definition A.1 is purely syntactic in nature. We will always work under the assumption that the obvious checks needed to ensure a semantically correct execution of a public-key encryption scheme Π will be carried out directly by the algorithms defining Π. For example, given all the appropriate inputs, we assume that the algorithm Enc (resp. Dec) itself checks if the tuple pp of public parameters is well-formed, if $m \in \mathcal{M}$ (resp. $c \in \mathcal{C}$), and if the key pk (resp. pk and sk) is compatible with pp before actually performing the encryption (resp. decryption), and that it proceeds accordingly (e.g., outputting $\perp$) should this condition not be satisfied. Similar remarks hold for the other algorithms.

(Perfect) Correctness. Intuitively, the correctness property of a public-key encryption scheme captures the idea that the algorithms work and interact properly if used as intended. Concretely this means that, whenever Bob fixes a security parameter $\lambda \in \mathbb{N}$ and generates a tuple of public parameters pp and a key pair $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K}$, then Alice encrypts a message $m \in \mathcal{M}$ using pk, and afterwards Bob decrypts with sk the ciphertext sent by Alice, both parties will obtain valid outputs and Bob will recover the same message m Alice had started with. This notion comes in two flavors: In the case of perfect correctness, the above is guaranteed to be true for every $m \in \mathcal{M}$ and every choice of the random coins used during the execution of the algorithms. On the other hand, (regular) correctness allows for a negligible error probability at each step. The formal definition is given below.

Definition A.2 ((Perfect) Correctness) *Let* $\Pi := (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme. We say that* Π *is* correct *(resp.* perfectly correct*) if:*

- *for every security parameter $\lambda \in \mathbb{N}$ and all public parameters* $\mathsf{pp} \leftarrow_{\$} \mathsf{PGen}(1^{\lambda})$,
- *for all key pairs* $(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{KGen}(\mathsf{pp})$,
- *for every message $m \in \mathcal{M}$,*

- *and for all ciphertexts* $c \leftarrow_{\$} \mathsf{Enc}(\mathsf{pp}, m, \mathsf{pk})$,

if $m' \leftarrow \mathsf{Dec}(\mathsf{pp}, c, \mathsf{pk}, \mathsf{sk})$*, then* $\mathbb{P}[m' \neq m] = \mathsf{negl}(\lambda)$ *(resp.* $\mathbb{P}[m' \neq m] = 0$*), the probability being taken over the random coins used by all the above algorithms.*

IND−CPA- and IND−CCA2-Security. Security of a public-key encryption scheme is captured by the notion of *indistinguishability.* It requires that any efficient adversary $\mathcal{A}$ with access to an encryption of a message $m \in \mathcal{M}$ and the length of m be able to determine any information on m with probability only negligibly higher than any other efficient adversary that only has access to the length of m. In other words, the knowledge of an encryption of m (in addition to the length of m) should not reveal any further information that can be extracted by an efficient adversary. This notion was first put forward in this form (sometimes called *semantic security*) by Goldwasser *et al.* (see [GM82]), and afterwards shown to be equivalent to the one we describe here by Goldwasser *et al.* in [GM84].

Indistinguishability is defined by the following security experiment. First, a key pair is generated and the public key is given to an efficient adversary $\mathcal{A}$, who may use it to create any polynomial number of ciphertexts. $\mathcal{A}$ then generates two messages $m_0, m_1 \in \mathcal{M}$ of the same length and forwards them to a challenge oracle, which picks one at random, encrypts it under the public key, and hands the result back to the adversary. The goal of $\mathcal{A}$ is to determine which of the two messages was chosen by the oracle. The scheme Π has indistinguishable encryptions under chosen-plaintext attack (or is IND−CPA-secure) if no such adversary succeeds in doing so with probability significantly greater than 1/2.

It is also possible to define variants of this notion where, in addition to the public key, $\mathcal{A}$ is given access to a decryption oracle which decrypts arbitrary ciphertexts at $\mathcal{A}$'s request. In the definition of indistinguishability under chosen ciphertext attack (IND−CCA1-security), $\mathcal{A}$ is allowed to query such a decryption oracle only up until it generates the two messages m_0, m_1. On the other hand, in the definition of indistinguishability under adaptive chosen ciphertext attack (IND−CCA2-security), $\mathcal{A}$ may query the decryption oracle even after it has forwarded the two messages, albeit without asking for a decryption of the challenge ciphertext (without this restriction the notion would be trivially unachievable). The formal definition of all these security notions is given below.

Definition A.3 (IND−T-Security) *Let* $\Pi := (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme,* $T \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$*, and* $\lambda \in \mathbb{N}$ *be a security parameter. Consider an efficient two-stage adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, where:*

$\mathcal{A}_1$*:*
- *a) Takes as input a tuple of public parameters* pp *corresponding to the security parameter* λ *and a public encryption key* pk*;*
- *b) Has access to a decryption oracle* $\mathcal{O}^{\mathsf{Dec}}$*, except if* $T = \mathsf{CPA}$*, in which case it is allowed no oracle queries (modeled here by the trival oracle* $\mathcal{O}^{\mathsf{Dec}}$ *mapping every input to* $\perp$*);*
- *c) Returns two messages* $m_0, m_1 \in \mathcal{M}$ *of the same length and some state information* st*;*

$\mathcal{A}_2$*:*
- *a) Takes as input a tuple of public parameters* pp *corresponding to the security parameter* λ*, a public encryption key* pk*, a ciphertext* $c^* \in \mathcal{C}$*, and some state information* st*;*
- *b) Is allowed no oracle queries (modeled here by the trivial oracle* $\tilde{\mathcal{O}}^{\mathsf{Dec}}$ *mapping every input to* $\perp$*), except if* $T = \mathsf{CCA2}$*, in which case it has access to a restricted decryption oracle* $\tilde{\mathcal{O}}^{\mathsf{Dec}}$*;*
- *c) Returns a bit* $b^* \in \{0, 1\}$*.*

For every such adversary $\mathcal{A}$*, let* $\mathbf{Exp}_{\mathcal{A},\Pi}^{\mathsf{IND}-T}(\lambda)$ *be the experiment defined in Figure 6. We say that* Π *is* IND−T-secure *if, for every adversary* $\mathcal{A}$ *as above,* $\mathbf{Adv}_{\mathcal{A},\Pi}^{\mathsf{IND}-T}(\lambda) = \mathbb{P}\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{\mathsf{IND}-T}(\lambda) = 1\right] - 1/2 = \mathsf{negl}(\lambda)$*, where the probability is taken over the random coins used by* $\mathcal{A}$*, as well as the random coins used in the experiment.*

```
$\mathbf{Exp}^{\mathsf{IND}-T}_{\mathcal{A},\Pi}(\lambda)$:
$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathsf{KGen}(\mathsf{pp})$
$b \leftarrow_\$ \{0,1\}$
$(m_0, m_1, \mathsf{st}) \leftarrow_\$ \mathcal{A}_1^{\mathcal{O}^{\mathsf{Dec}}}(\mathsf{pp}, \mathsf{pk})$
if $m_0 \notin \mathcal{M} \ \vee \ m_1 \notin \mathcal{M} \ \vee \ |m_0| \neq |m_1|$ then
    return 0
$c^* \leftarrow_\$ \mathsf{Enc}(\mathsf{pp}, m_b, \mathsf{pk})$
$b^* \leftarrow_\$ \mathcal{A}_2^{\tilde{\mathcal{O}}^{\mathsf{Dec}}}(\mathsf{pp}, \mathsf{pk}, c^*, \mathsf{st})$
if $b = b^*$ then
    return 1
return 0
```

```
$\mathcal{O}^{\mathsf{Dec}}(c)$:
if $T = \mathsf{CPA}$ then
    return $\perp$
$m \leftarrow \mathsf{Dec}(\mathsf{pp}, c, \mathsf{pk}, \mathsf{sk})$
return $m$
```

```
$\tilde{\mathcal{O}}^{\mathsf{Dec}}(c)$:
if $T \in \{\mathsf{CPA}, \mathsf{CCA1}\} \ \vee \ c = c^*$ then
    return $\perp$
$m \leftarrow \mathsf{Dec}(\mathsf{pp}, c, \mathsf{pk}, \mathsf{sk})$
return $m$
```

Figure 6: $\mathsf{IND}-T$-security, $T \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$

A.2 Digital Signature Schemes

Definition A.4 (Digital Signature Schemes) *A* digital signature scheme Σ *is a tuple of four probabilistic polynomial-time algorithms* $\Sigma := (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ *defined as follows:*

$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$: *On input a security parameter $\lambda \in \mathbb{N}$, the algorithm* PGen *generates a tuple of public parameters of Σ. The tuple* pp *includes the security parameter 1^λ, the definition of the message space $\mathcal{M}$, the space of signatures $\mathcal{S}$, and the key space $\mathcal{K}$ (together with two special symbols $\top, \perp \notin \mathcal{M} \cup \mathcal{S} \cup \mathcal{K}$), along with any other information needed to sign messages or verify signatures, except for identities and user key pairs.*

$(\mathsf{vk}, \mathsf{sk}) \leftarrow_\$ \mathsf{KGen}(\mathsf{pp})$: *On input a tuple of public parameters* pp*, the algorithm* KGen *returns a key pair or an error message, that is $(\mathsf{vk}, \mathsf{sk}) \in \mathcal{K} \cup \{\perp\}$. Here,* vk *and* sk *are the public verification and the signer secret keys, respectively. We write $\mathcal{K}_{\mathsf{vk}}$ and $\mathcal{K}_{\mathsf{sk}}$ for the sets of all possible public verification and signer secret keys.*

$\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}, \mathsf{vk})$: *On input a tuple of public parameters* pp*, a message $m \in \mathcal{M}$, a signer secret key* $\mathsf{sk} \in \mathcal{K}_{\mathsf{sk}}$*, and a public verification key* $\mathsf{vk} \in \mathcal{K}_{\mathsf{vk}}$*, the algorithm* Sign *returns a signature or an error message, that is $\sigma \in \mathcal{S} \cup \{\perp\}$.*

$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m, \sigma, \mathsf{vk})$: *On input a tuple of public parameters* pp*, a message $m \in \mathcal{M}$, a signature $\sigma \in \mathcal{S}$, and a public verification key* $\mathsf{vk} \in \mathcal{K}_{\mathsf{vk}}$*, the deterministic algorithm* Verify *returns a bit $d \in \{\top, \perp\}$.*

Remark. Observe that Definition A.4 is purely syntactic in nature. We will always assume that the obvious checks needed to ensure a semantically correct execution of a digital signature scheme Σ will be carried out directly by the algorithms defining Σ. For example, given all the appropriate inputs, we assume that the algorithm Sign itself checks if the tuple pp of public parameters is well-formed, if $m \in \mathcal{M}$, and if the keys pk and sk are compatible with pp before producing the signature, and that it proceeds accordingly (e.g., outputting $\perp$) should this condition not be satisfied. Similar remarks hold for the other algorithms.

(Perfect) Correctness. Again, the intuition behind the correctness property of a digital signature scheme is that the algorithms work and interact properly if used as intended. Concretely this means that, whenever Alice fixes a security parameter $\lambda \in \mathbb{N}$, generates a tuple of public parameters pp and a key pair $(\mathsf{vk}, \mathsf{sk}) \in \mathcal{K}$, signs a message $m \in \mathcal{M}$ using sk, and then Bob verifies Alice's signature with vk, both parties will obtain valid outputs and Bob will conclude that the signature is valid. As before, the correctness property can be defined by requiring the above to hold for every $m \in \mathcal{M}$ and every choice of the random coins used during the execution of the algorithms (perfect correctness), or by allowing for a negligible error probability at each step ((regular) correctness). The formal definition is given below.

Definition A.5 ((Perfect) Correctness) *Let* $\Sigma := (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a digital signature scheme. We say that* Σ *is* correct *(resp.* perfectly correct*) if:*

- *for every security parameter* $\lambda \in \mathbb{N}$ *and all public parameters* $\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$*,*
- *for all key pairs* $(\mathsf{vk}, \mathsf{sk}) \leftarrow_\$ \mathsf{KGen}(\mathsf{pp})$*,*
- *for every message* $m \in \mathcal{M}$*,*
- *and for all signatures* $\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}, \mathsf{vk})$*,*

if $d \leftarrow \mathsf{Verify}(\mathsf{pp}, m, \sigma, \mathsf{vk})$*, then* $\mathbb{P}[\, d = \bot] = \mathsf{negl}(\lambda)$ *(resp.* $\mathbb{P}[\, d = \bot] = 0$*), the probability being taken over the random coins used by all the above algorithms.*

(Strong) Unforgeability. The most important security property of a digital signature scheme Σ is *unforgeability*: No efficient adversary $\mathcal{A}$ should be able to generate a valid signature of a message not previously endorsed by the legitimate signer. It was first introduced by Goldwasser *et al.* in [GMR88].

Unforgeability can be defined by means of a security experiment. First, a key pair is generated and the verification key is given to an efficient adversary $\mathcal{A}$, who has full adaptive access to a signing oracle. $\mathcal{A}$'s goal is to generate a valid signature of any message not already queried to the oracle. Σ is unforgeable if any such adversary succeeds in doing so only with negligible probability.

While this definition appears to be sound, it has some slight shortcomings. For example, it does not prevent $\mathcal{A}$ from deriving fresh signatures of already queried messages, as the winning condition in the unforgeability experiment only requires the adversary output a signature of a message never queried before. But once $\mathcal{A}$ has learned one signature generated from a specific message, he could potentially forge many other signatures of the same message—a problem which may have unwanted consequences in many practical applications.

Hence, it is reasonable to alter the definition so that $\mathcal{A}$ cannot generate any fresh signature on his own. This is the approach taken by An *et al.* in [ADR02]: The idea is to keep track not only of the messages queried to the oracle, but also of the answers, and to demand that $\mathcal{A}$ come up with a completely new valid message-signature pair. This approach leads to the definition of strong unforgeability: In the strong unforgeability experiment $\mathcal{A}$ wins if he can generate any valid message-signature pair not already seen before. The formal definition of (strong) unforgeability is given below.

Definition A.6 ((Strong) Unforgeability) *Let* $\Sigma := (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a digital signature scheme and* $\lambda \in \mathbb{N}$ *be a security parameter. Consider an efficient adversary* $\mathcal{A}$ *which:*

a) Takes as input a tuple of public parameters pp *corresponding to the security parameter* λ *and a public verification key* vk*;*
b) Has access to a signing oracle $\mathcal{O}^{\mathsf{Sign}}$ *(resp.* $\tilde{\mathcal{O}}^{\mathsf{Sign}}$*);*
c) Returns a pair $(m^*, \sigma^*) \in \mathcal{M} \times \mathcal{S}$*.*

For every such adversary $\mathcal{A}$*, let* $\mathbf{Exp}^{T}_{\mathcal{A},\Sigma}(\lambda)$ *with* $T = \mathsf{Unf}$ *(resp.* $T = \mathsf{SUnf}$*) be the experiment defined in the left-hand side (resp. right-hand side) of Figure 7. We say that* Σ *is* unforgeable under adaptive chosen-message attack *(resp.* strongly unforgeable under adaptive chosen-message attack*) if, for every adversary* $\mathcal{A}$ *as above,* $\mathbf{Adv}^{T}_{\mathcal{A},\Sigma}(\lambda) = \mathbb{P}\Big[\mathbf{Exp}^{T}_{\mathcal{A},\Sigma}(\lambda) = 1\Big] = \mathsf{negl}(\lambda)$*, where the probability is taken over the random coins used by* $\mathcal{A}$*, as well as the random coins used in the experiment.*

$\mathbf{Exp}^{\mathsf{Unf}}_{\mathcal{A},\Sigma}(\lambda)$:

```
$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
$(\mathsf{vk}, \mathsf{sk}) \leftarrow_\$ \mathsf{KGen}(\mathsf{pp})$
$B \leftarrow \emptyset$
$(m^*, \sigma^*) \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}}}(\mathsf{pp}, \mathsf{vk})$
$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m^*, \sigma^*, \mathsf{vk})$
if $d = \top \ \wedge \ m^* \notin B$ then
    return 1
return 0
```

$\mathcal{O}^{\mathsf{Sign}}(m)$:

```
$\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}, \mathsf{vk})$
$B \leftarrow B \cup \{m\}$
return $\sigma$
```

$\mathbf{Exp}^{\mathsf{SUnf}}_{\mathcal{A},\Sigma}(\lambda)$:

```
$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
$(\mathsf{vk}, \mathsf{sk}) \leftarrow_\$ \mathsf{KGen}(\mathsf{pp})$
$B \leftarrow \emptyset$
$(m^*, \sigma^*) \leftarrow_\$ \mathcal{A}^{\tilde{\mathcal{O}}^{\mathsf{Sign}}}(\mathsf{pp}, \mathsf{vk})$
$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m^*, \sigma^*, \mathsf{vk})$
if $d = \top \ \wedge \ (m^*, \sigma^*) \notin B$ then
    return 1
return 0
```

$\tilde{\mathcal{O}}^{\mathsf{Sign}}(m)$:

```
$\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}, \mathsf{vk})$
$B \leftarrow B \cup \{(m, \sigma)\}$
return $\sigma$
```

Figure 7: Unforgeability (left) and strong unforgeability (right)

B Correctness of Sanitizable Signature Schemes

In this Appendix we expand on the correctness definitions of sanitizable signature schemes that we have briefly touched upon in Section 3.3. In doing so, we take inspiration from the equivalent notion for (ordinary) digital signature schemes presented in Appendix A.2. Again, the intuitive idea behind the definitions is that all the algorithms work and interact properly if used as intended. Compared to the case of digital signature schemes, however, the main difference here is that there are many more algorithms in a sanitizable signature scheme—in particular, there are two additional parties, the sanitizer and the judge, who do not have a counterpart in the "ordinary" setting. As a consequence, there are also more correctness notions.

As usual, every one of these concepts may be of two types: In the perfect version, the requirements must hold for every choice made by the parties involved and every choice of the random coins used during the execution of the algorithms. The "regular" version on the other hand allows for a negligible error probability at each step.

Of the three properties discussed here, only the first was already introduce by Ateniese *et al.* in [ACdT05]; the other two properties later appeared in [BFF⁺09].

(Perfect) Signing Correctness. The first notion deals with correctness from the signer's point of view. It states that, whenever a singer and a sanitizer key pair are generated and the signer signs a message delegating certain sanitizing rights to the sanitizer, both parties will obtain valid outputs and the signature will also be valid under the corresponding public keys. The formal definition is given below.

Definition B.1 ((Perfect) Signing Correctness) *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme. We say that* SSS *is* signing correct *(resp.* perfectly signing correct*) if:*

- *for every security parameter* $\lambda \in \mathbb{N}$ *and all public parameters* $\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$*,*
- *for all key pairs* $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$*,* $(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$*,*
- *for every message* $m \in \mathcal{M}$*,*
- *for all sanitizing rights* ADM *with* $\mathrm{ADM}(m) = \top$*,*
- *and for every* $\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM})$*,*

if $d \leftarrow \mathsf{Verify}(\mathsf{pp}, m, \sigma, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$*, then* $\mathbb{P}[\, d = \bot] = \mathsf{negl}(\lambda)$ *(resp.* $\mathbb{P}[\, d = \bot] = 0$*), the probability being taken over the random coins used by all the above algorithms.*

(Perfect) Sanitizing Correctness. The second notion deals with correctness from the sanitizer's point of view. It states that, whenever a singer and a sanitizer key pair are generated, the signer signs a message $m \in \mathcal{M}$ delegating certain sanitizing rights to the sanitizer, and the sanitizer modifies m within the limits set by the signer and generates a sanitized signature for the modified message, both parties will obtain valid outputs and the signature will also be valid under the corresponding public keys. The formal definition is given below.

Definition B.2 ((Perfect) Sanitizing Correctness) *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme. We say that* SSS *is* sanitizing correct *(resp.* perfectly sanitizing correct*) if:*

- *for every security parameter* $\lambda \in \mathbb{N}$ *and all public parameters* $\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$*,*
- *for all key pairs* $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$*,* $(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$*,*
- *for every message* $m \in \mathcal{M}$*,*
- *for all sanitizing rights* ADM *with* $\mathrm{ADM}(m) = \top$*,*
- *for every* $\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM})$*,*

- *for all sanitizing instructions* MOD *with* $\mathrm{MOD}(m) \neq \bot$ *and* $\mathrm{MOD}(\mathrm{ADM}) = \top$,
- *and for every* $\sigma' \leftarrow_\$ \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD})$,

if $d \leftarrow \mathsf{Verify}(\mathsf{pp}, m', \sigma', \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$, *then* $\mathbb{P}[\, d = \bot] = \mathsf{negl}(\lambda)$ *(resp.* $\mathbb{P}[\, d = \bot] = 0$*), the probability being taken over the random coins used by all the above algorithms.*

(Perfect) Proof Correctness. The final notion deals with correctness from the perspective of the judge. It states that if a signature is correctly generated in one of the two ways discussed above, and if the signer uses his secret key to generate a proof aimed at resolving an authorship dispute about that particular signature, both parties will obtain valid outputs and the judge will ascribe the signature to the correct party. The formal definition is given below.

Definition B.3 ((Perfect) Proof Correctness) *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme. We say that* SSS *is* proof correct *(resp.* perfectly proof correct*) if:*

- *for every security parameter* $\lambda \in \mathbb{N}$ *and all public parameters* $\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$,
- *for all key pairs* $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$, $(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$,
- *for every message* $m \in \mathcal{M}$,
- *for all sanitizing rights* ADM *with* $\mathrm{ADM}(m) = \top$
- *for every* $\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM})$,
- *for all sanitizing instructions* MOD *with* $\mathrm{MOD}(m) \neq \bot$ *and* $\mathrm{MOD}(\mathrm{ADM}) = \top$,
- *for every* $\sigma' \leftarrow_\$ \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD})$,
- *for all messages* $m = m_1, m_2, \ldots, m_k \in \mathcal{M}$,
- *for every sanitizing rights* $\mathrm{ADM} = \mathrm{ADM}_1, \mathrm{ADM}_2, \ldots, \mathrm{ADM}_k$ *with* $\mathrm{ADM}_i(m_i) = \top$, $2 \leq i \leq k$,
- *for all signatures* $\sigma_i \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m_i, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM}_i)$, $2 \leq i \leq k$,
- *and for all proofs*

$$\pi \leftarrow_\$ \mathsf{Proof}(\mathsf{pp}, m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}), \quad \pi' \leftarrow_\$ \mathsf{Proof}(\mathsf{pp}, m', \sigma', \{(m_i, \sigma_i)\}_{i=1}^k, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}),$$

if $d \leftarrow \mathsf{Judge}(\mathsf{pp}, m, \sigma, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \pi)$ *and* $d' \leftarrow \mathsf{Judge}(\mathsf{pp}, m', \sigma', \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \pi)$, *then* $\mathbb{P}[\, d \neq \mathsf{Sig}] = \mathsf{negl}(\lambda)$ *and* $\mathbb{P}[\, d' \neq \mathsf{San}] = \mathsf{negl}(\lambda)$ *(resp.* $\mathbb{P}[\, d \neq \mathsf{Sig}] = 0$ *and* $\mathbb{P}[\, d' \neq \mathsf{San}] = 0$*), the probability being taken over the random coins used by all the above algorithms.*

C Security Definitions for Sanitizable Signature Schemes

In this Appendix we elaborate on the security definitions of sanitizable signature schemes that we have outlined in Section 3.3. For each of these notions we give some background information and then define it rigorously by means of a security experiment.

(Strong) Unforgeability. Given that a sanitizable signature scheme SSS allows legitimate users to sign or sanitize messages, the most basic property SSS should satisfy is *unforgeability*: Similarly to digital signature schemes, it requires that no efficient adversary $\mathcal{A}$ be able to generate a valid signature of a message not previously endorsed by the signer or the sanitizer. This property is the analogous of unforgeability for digital signature schemes (see Appendix A.2).

Unforgeability was already introduced and formalized in a game-based fashion by Ateniese *et al.* in [ACdT05] and later included by Brzuska *et al.* [BFF$^+$09] in their security model. This initial definition was shown to be slightly inappropriate by Gong *et al.* [GQZ11] (it does not protect against rights forge attacks), who proposed an adjustment to solve this problem (also see de Meer *et al.* [dMPPS14]). The notion has since been further stengthened by Krenn *et al.* [KSS16] to account also for signatures (and not only for messages and sanitizing rights, thus in particular adressing the concerns raised in [GQZ11]) and was later adopted in this stronger form by Camenisch *et al.* [CDK$^+$17] and Beck *et al.* [BCD$^+$17].

Unforgeability can be defined by means of a security experiment. First, a signer and a sanitizer key pair are generated and given to an efficient adversary $\mathcal{A}$, who has full adaptive oracle access. $\mathcal{A}$'s goal is to generate a valid signature of a message not already signed or sanitized by any legitimate party (i.e., not already queried to $\mathcal{O}^{\mathsf{Sign}}$ or $\mathcal{O}^{\mathsf{Sanit}}$). SSS is unforgeable if any such adversary succeeds in doing so only with negligible probability.

This definition mimics the classical definition of (standard) unforgeability for digital signature schemes given in [GMR88] and thus also suffers from the same shortcomings. In particular, it does not prevent $\mathcal{A}$ from deriving fresh signatures of already queried messages, with potentially different sanitizing rights than those originally assigned by the honest signer.

Hence, it is reasonable to again alter the definition so that $\mathcal{A}$ cannot generate any fresh signature on his own. This can be done by following the same approach as in Appendix A.2, which results in $\mathcal{A}$ already winning if he can generate any valid message-signature pair not seen before. This is the idea behind the notion of strong unforgeability of a sanitizable signature scheme.
We conclude by observing that this security property takes into account only adversaries acting from "outside" the system; security properties dealing with malicious signers or sanitizers will be introduced in the following paragraphs. The formal definition of (strong) unforgeability is given below.

Definition C.1 ((Strong) Unforgeability) *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme and* $\lambda \in \mathbb{N}$ *be a security parameter. Consider an efficient adversary* $\mathcal{A}$ *which:*

a) Takes as input a tuple of public parameters pp *corresponding to the security parameter* λ*, a signer public key* $\mathsf{pk}_{\mathsf{Sig}}$*, and a sanitizer public key* $\mathsf{pk}_{\mathsf{San}}$*;*

b) Has access to a signing oracle $\mathcal{O}^{\mathsf{Sign}}$ *(resp.* $\tilde{\mathcal{O}}^{\mathsf{Sign}}$*), a sanitizing oracle* $\mathcal{O}^{\mathsf{Sanit}}$ *(resp.* $\tilde{\mathcal{O}}^{\mathsf{Sanit}}$*), and a proof oracle* $\mathcal{O}^{\mathsf{Proof}}$*;*

c) Returns a pair $(m^*, \sigma^*) \in \mathcal{M} \times \mathcal{S}$*.*

For every such adversary $\mathcal{A}$*, let* $\mathbf{Exp}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda)$ *with* $T = \mathsf{Unf}$ *(resp.* $T = \mathsf{SUnf}$*) be the experiment defined in the upper-left corner (resp. on the left) of Figure 8. We say that* SSS *is* unforgeable *(resp.* strongly unforgeable*) if, for every efficient adversary* $\mathcal{A}$ *as above,* $\mathbf{Adv}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda) = \mathbb{P}\left[\mathbf{Exp}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda) = 1\right] = \mathsf{negl}(\lambda)$*, where the probability is taken over the random coins used by* $\mathcal{A}$*, as well as the random coins used in the experiment.*

$\mathbf{Exp}^{\mathsf{Unf}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:

```
$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
$(\mathsf{pk}_\mathsf{Sig}, \mathsf{sk}_\mathsf{Sig}) \leftarrow_\$ \mathsf{KGen}_\mathsf{Sig}(\mathsf{pp})$
$(\mathsf{pk}_\mathsf{San}, \mathsf{sk}_\mathsf{San}) \leftarrow_\$ \mathsf{KGen}_\mathsf{San}(\mathsf{pp})$
$B_\mathsf{Sign} \leftarrow \emptyset,\ B_\mathsf{Sanit} \leftarrow \emptyset$
$(m^*, \sigma^*) \leftarrow_\$ \mathcal{A}^{\mathcal{O}^\mathsf{Sign}, \mathcal{O}^\mathsf{Sanit}, \mathcal{O}^\mathsf{Proof}}(\mathsf{pp}, \mathsf{pk}_\mathsf{Sig}, \mathsf{pk}_\mathsf{San})$
$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}_\mathsf{Sig}, \mathsf{pk}_\mathsf{San})$
if $d = \top \ \wedge \ (\mathsf{pk}_\mathsf{San}, m^*) \notin B_\mathsf{Sign} \ \wedge$
    $(\mathsf{pk}_\mathsf{Sig}, m^*) \notin B_\mathsf{Sanit}$ then
    return 1
return 0
```

$\mathcal{O}^\mathsf{Sign}(m, \mathsf{pk}'_\mathsf{San}, \mathrm{ADM})$:

```
$\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_\mathsf{Sig}, \mathsf{pk}_\mathsf{Sig}, \mathsf{pk}'_\mathsf{San}, \mathrm{ADM})$
$B_\mathsf{Sign} \leftarrow B_\mathsf{Sign} \cup \{(\mathsf{pk}'_\mathsf{San}, m)\}$
return $\sigma$
```

$\mathcal{O}^\mathsf{Sanit}\Big(m, \sigma, \mathsf{pk}'_\mathsf{Sig}, \mathrm{MOD}\Big)$:

```
$m' \leftarrow \mathrm{MOD}(m)$
$\sigma' \leftarrow_\$ \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_\mathsf{San}, \mathsf{pk}'_\mathsf{Sig}, \mathsf{pk}_\mathsf{San}, \mathrm{MOD})$
$B_\mathsf{Sanit} \leftarrow B_\mathsf{Sanit} \cup \{(\mathsf{pk}'_\mathsf{Sig}, m')\}$
return $\sigma'$
```

$\mathbf{Exp}^{\mathsf{SUnf}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:

```
$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
$(\mathsf{pk}_\mathsf{Sig}, \mathsf{sk}_\mathsf{Sig}) \leftarrow_\$ \mathsf{KGen}_\mathsf{Sig}(\mathsf{pp})$
$(\mathsf{pk}_\mathsf{San}, \mathsf{sk}_\mathsf{San}) \leftarrow_\$ \mathsf{KGen}_\mathsf{San}(\mathsf{pp})$
$B_\mathsf{Sign} \leftarrow \emptyset,\ B_\mathsf{Sanit} \leftarrow \emptyset$
$(m^*, \sigma^*) \leftarrow_\$ \mathcal{A}^{\tilde{\mathcal{O}}^\mathsf{Sign}, \tilde{\mathcal{O}}^\mathsf{Sanit}, \mathcal{O}^\mathsf{Proof}}(\mathsf{pp}, \mathsf{pk}_\mathsf{Sig}, \mathsf{pk}_\mathsf{San})$
$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}_\mathsf{Sig}, \mathsf{pk}_\mathsf{San})$
if $d = \top \ \wedge \ (\mathsf{pk}_\mathsf{San}, m^*, \sigma^*) \notin B_\mathsf{Sign} \ \wedge$
    $(\mathsf{pk}_\mathsf{Sig}, m^*, \sigma^*) \notin B_\mathsf{Sanit}$ then
    return 1
return 0
```

$\tilde{\mathcal{O}}^\mathsf{Sign}(m, \mathsf{pk}'_\mathsf{San}, \mathrm{ADM})$:

```
$\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_\mathsf{Sig}, \mathsf{pk}_\mathsf{Sig}, \mathsf{pk}'_\mathsf{San}, \mathrm{ADM})$
$B_\mathsf{Sign} \leftarrow B_\mathsf{Sign} \cup \{(\mathsf{pk}'_\mathsf{San}, m, \sigma)\}$
return $\sigma$
```

$\tilde{\mathcal{O}}^\mathsf{Sanit}\Big(m, \sigma, \mathsf{pk}'_\mathsf{Sig}, \mathrm{MOD}\Big)$:

```
$m' \leftarrow \mathrm{MOD}(m)$
$\sigma' \leftarrow_\$ \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_\mathsf{San}, \mathsf{pk}'_\mathsf{Sig}, \mathsf{pk}_\mathsf{San}, \mathrm{MOD})$
$B_\mathsf{Sanit} \leftarrow B_\mathsf{Sanit} \cup \{(\mathsf{pk}'_\mathsf{Sig}, m', \sigma')\}$
return $\sigma'$
```

$\mathcal{O}^\mathsf{Proof}\Big(m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, \mathsf{pk}'_\mathsf{San}\Big)$:

```
$\pi \leftarrow_\$ \mathsf{Proof}(\mathsf{pp}, m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, \mathsf{sk}_\mathsf{Sig}, \mathsf{pk}'_\mathsf{San})$
return $\pi$
```

Figure 8: Unforgeability (left) and strong unforgeability (right)

Immutability. Sanitizable signature schemes allow the signer to declare certain message blocks as admissible for modification, so that a pre-determined sanitizer is later able to edit those blocks and to produce a valid signature that verifies under the signer and sanitizer public keys. On the other hand, non-admissible message blocks should be immutable for everyone: Neither an external entity, nor the intended sanitizer should be able to modify those blocks and to produce a valid signature. The former case has already been addressed in the previous paragrah. The *immutability* property is concerned with the latter: A malicious sanitizer should not be able to change non-admissible message blocks and to produce a valid signature (even holding his own private key).

Immutability, too, has already been introduced by Ateniese *et al.* in [ACdT05], but was formalized for the first time later by Brzuska *et al.* [BFF$^+$09]. The definition of immutability has then been largely adopted by subsequent authors in this orignal form (see e.g. [BCD$^+$17, BFLS10, BPS12, CDK$^+$17, KSS16]), with only some minor exceptions (see de Meer *et al.* [dMPPS14], who define a weaker notion of immutability, where the adversary knows, but does not generate, the sanitizer key pair).

Immutability can be defined by means of a security experiment. First, a signer key pair is generated and the public key is given to an efficient adversary $\mathcal{A}$, who has full adaptive oracle access. $\mathcal{A}$'s goal is to impersonate a sanitizer (i.e., to output a sanitizer public key $\mathsf{pk}^*_{\mathsf{San}}$) and to generate a valid message-signature pair that is not an admissible modification of a message previously signed by the signing oracle. SSS is immutable if any such adversary succeeds in doing so only with negligible probability.

Observe that, at first sight, it seems that this notion does not protect against rights forge attacks mounted by a malicious sanitizer (which are not covered by unforgeability, since a malicious sanitizer has the additional knowledge of the sanitizer secret key). Indeed, in this setting the adversary is not barred from producing a valid signature, equipped with different sanitizing rights, of a previously queried message or a legally modified message. In other words, he is allowed to "sanitize" a message by keeping it unmodified or applying matching modifications, and then encoding a different set of admissible blocks into the signature.

In [GQZ11] the problem of rights forge attacks is addressed by keeping track of all the sanitizing rights queried to the oracle, and then resorting the same trick we have used in the definition of strong unforgeability: One of the winning conditions is already satisfied if the triple $(\mathsf{pk}^*_{\mathsf{San}}, m^*, \sigma^*)$ returned by $\mathcal{A}$ is such that $(\mathsf{pk}^*_{\mathsf{San}}, m^*, \mathrm{ADM}^*)$ has not been queried before to the signing oracle (ADM^* being the sanitizing rights encoded into σ^*). This solution turns out to be satisfactory, because they work under the critical assumption that the sanitizing rights encoded into a signature are always publicly recoverable, which we do not have in our setting. Regrettably, we therefore cannot use the same strategy here.

On the other hand, we observe that a rights forge attack can essentially be of two types: Let $m \in \mathcal{M}$ be the message signed by the signer, let A be the set of block indexes declared as admissible by the signer, and let A' be the set of index blocks encoded as admissible in the signature produced by the malicious sanitizer. In order for there to be a rights forge attack we must have $A \neq A'$, which means that either $A' \nsubseteq A$ or $A' \subsetneq A$. But observe that a malicious sanitizer that can mount a rights forge attack of the first kind can easily be turned into an adversary violating immutability (this adversary simply has to modify the newly won admissible block), whereas a rights forge attack of the second type seems unlikely to be in the interest of any malicious sanitizer, becuase it effectively downgrades his modification rights. In this respect, rights-forge attacks mounted by a malicious sanitizer are already covered to a certain extent by notion of immutability. However, we acknowledge that the possibility of restricting modification rights could potentially be at odds with accountability requirements, which is something that deserves further investigation. The formal definition of immutability is given below.

Definition C.2 (Immutability) *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme and* $\lambda \in \mathbb{N}$ *be a security parameter. Consider an efficient adversary* $\mathcal{A}$ *which:*

a) Takes as input a tuple of public parameters pp *corresponding to the security parameter* λ *and a signer public key* $\mathsf{pk}_{\mathsf{Sig}}$*;*

b) Has access to a signing oracle $\mathcal{O}^{\mathsf{Sign}}$ *and a proof oracle* $\mathcal{O}^{\mathsf{Proof}}$*;*

c) Returns a tuple $(m^*, \sigma^*, \mathsf{pk}^*_{\mathsf{San}}) \in \mathcal{M} \times \mathcal{S} \times \mathcal{K}_{\mathsf{San,pk}}$*.*

For every such adversary $\mathcal{A}$*, let* $\mathbf{Exp}^{\mathsf{Imm}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$ *be the experiment defined in Figure 9. We say that* SSS *is* immutable *if, for every efficient adversary* $\mathcal{A}$ *as above,* $\mathbf{Adv}^{\mathsf{Imm}}_{\mathcal{A},\mathsf{SSS}}(\lambda) = \mathbb{P}\left[\mathbf{Exp}^{\mathsf{Imm}}_{\mathcal{A},\mathsf{SSS}}(\lambda) = 1\right] = \mathsf{negl}(\lambda)$*, where the probability is taken over the random coins used by* $\mathcal{A}$*, as well as the random coins used in the experiment.*

(Strong) Privacy. A fundamental property of any sanitizable signature scheme is *privacy*: Once a message has been sanitized, an outsider should not be able to derive any information about the original content.

This property can be modelled by the following security experiment: First, a signer and a sanitizer key pair, as well as a secret bit $b \in \{0, 1\}$, are generated and the public keys are given to an adversary $\mathcal{A}$, who has

```
$\mathbf{Exp}^{\mathsf{Imm}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:
    $\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
    $(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$
    $B \leftarrow \emptyset$
    $(m^*, \sigma^*, \mathsf{pk}^*_{\mathsf{San}}) \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Proof}}}(\mathsf{pp}, \mathsf{pk}_{\mathsf{Sig}})$
    $d \leftarrow \mathsf{Verify}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}^*_{\mathsf{San}})$
    if $d = \top \ \wedge \ (\mathsf{pk}^*_{\mathsf{San}}, m^*) \notin B_{\mathrm{MOD}}$ then
        return 1
    return 0

$\mathcal{O}^{\mathsf{Sign}}(m, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM})$:
    $\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM})$
    $B \leftarrow B \cup \{(\mathsf{pk}_{\mathsf{San}}, m, \mathrm{ADM})\}$
    return $\sigma$

$\mathcal{O}^{\mathsf{Proof}}\Big(m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, \mathsf{pk}_{\mathsf{San}}\Big)$:
    $\pi \leftarrow_\$ \mathsf{Proof}(\mathsf{pp}, m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
    return $\pi$
```

Here $B_{\mathrm{MOD}} \coloneqq \{(\mathsf{pk}_{\mathsf{San}}, \mathrm{MOD}(m)) \in \mathcal{K}_{\mathsf{San,pk}} \times \mathcal{M} : (\mathsf{pk}_{\mathsf{San}}, m, \mathrm{ADM}) \in B, \mathrm{MOD} \in \mathcal{P}(\mathbb{N} \times \mathcal{M}) \times \mathbb{N}, \mathrm{MOD}(\mathrm{ADM}) = \top\}$.

Figure 9: Immutability

full adaptive oracle access. He is also allowed to query a left-or-right oracle $\mathcal{O}^{\mathsf{LoR}}$, where he can input two messages $m_0, m_1 \in \mathcal{M}$ (with the same description ADM) and two modifications $\mathrm{MOD}_0, \mathrm{MOD}_1$, with the only restriction that m_0 and m_1 get sanitized to the exact same message by the respective modifications. $\mathcal{O}^{\mathsf{LoR}}$ then signs m_b (making ADM admissible), and afterwards sanitizes the result according to MOD_b. $\mathcal{A}$'s goal is to determine which of the two initial messages the final (sanitized) signature comes from, that is to determine the bit b. SSS is private if no efficient adversary as above succeeds in doing so with probability significantly greater than 1/2.
This notion can be further strengthened by granting $\mathcal{A}$ access to the honestly generated sanitizer secret key. The formal definition of (strong) privacy is given below.

Definition C.3 ((Strong) Privacy) *Let* $\mathsf{SSS} \coloneqq (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme and* $\lambda \in \mathbb{N}$ *be a security parameter. Consider an efficient adversary* $\mathcal{A}$ *which:*

a) Takes as input a tuple of public parameters pp *corresponding to the security parameter* λ*, a signer public key* $\mathsf{pk}_{\mathsf{Sig}}$*, and a sanitizer public key* $\mathsf{pk}_{\mathsf{San}}$ *(as well as a sanitizer secret key* $\mathsf{sk}_{\mathsf{San}}$*, in case of strong transparency);*

b) Has access to a signing oracle $\mathcal{O}^{\mathsf{Sign}}$*, a sanitizing oracle* $\mathcal{O}^{\mathsf{Sanit}}$*, a proof oracle* $\mathcal{O}^{\mathsf{Proof}}$*, and a left-or-right oracle* $\mathcal{O}^{\mathsf{LoR}}$ *(in case of strong privacy, the oracle* $\mathcal{O}^{\mathsf{Sign}}$ *can be omitted);*

c) Returns a bit $b^* \in \{0,1\}$*.*

For every such adversary $\mathcal{A}$*, let* $\mathbf{Exp}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda)$ *with* $T = \mathsf{Priv}$ *(resp.* $T = \mathsf{SPriv}$*) be the experiment defined in the upper-left corner (resp. on the left) of Figure 10. We say that* SSS *is* private *(resp.* strongly private*) if, for every efficient adversary* $\mathcal{A}$ *as above,* $\mathbf{Adv}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda) = \mathbb{P}\Big[\mathbf{Exp}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda) = 1\Big] - 1/2 = \mathsf{negl}(\lambda)$*, where the probability is taken over the random coins used by* $\mathcal{A}$*, as well as the random coins used in the experiment.*

Transparency. A somewhat less common security property of sanitizable signature schemes is *transparency*, which is a strong privacy notion. In a nutshell, a sanitizable signature scheme is transparent if an outsider not holding any private keys is unable to decide whether a given signature has been generated by the signer or the sanitizer (as long as no proof to determine the accountable party for that message-signature pair has been published). In other words, the party accountable for any given signature remains hidden.

```
$\mathbf{Exp}^{\mathsf{Priv}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:
$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$
$(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$
$b \leftarrow_\$ \{0,1\}$
$b^* \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}},\mathcal{O}^{\mathsf{Sanit}},\mathcal{O}^{\mathsf{Proof}},\mathcal{O}^{\mathsf{LoR}}}(\mathsf{pp}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
if $b = b^*$ then
    return 1
return 0
```

```
$\mathcal{O}^{\mathsf{Sign}}(m, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM})$:
$\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM})$
return $\sigma$
```

```
$\mathcal{O}^{\mathsf{Sanit}}\Big(m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \mathrm{MOD}\Big)$:
$\sigma' \leftarrow_\$ \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}'_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD})$
return $\sigma'$
```

```
$\mathbf{Exp}^{\mathsf{SPriv}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:
$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$
$(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$
$b \leftarrow_\$ \{0,1\}$
$b^* \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}},\mathcal{O}^{\mathsf{Proof}},\mathcal{O}^{\mathsf{LoR}}}(\mathsf{pp}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}})$
if $b = b^*$ then
    return 1
return 0
```

```
$\mathcal{O}^{\mathsf{LoR}}(m_0, \mathrm{MOD}_0, m_1, \mathrm{MOD}_1, \mathrm{ADM})$:
if $\mathrm{ADM}(m_0) = \bot \ \vee\ \mathrm{ADM}(m_1) = \bot \ \vee$
  $\mathrm{MOD}_0(m_0) \neq \mathrm{MOD}_1(m_1)$ then
    return $\bot$
$\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m_b, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{ADM})$
$\sigma' \leftarrow_\$ \mathsf{Sanit}(\mathsf{pp}, m_b, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD}_b)$
return $\sigma'$
```

```
$\mathcal{O}^{\mathsf{Proof}}\Big(m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, \mathsf{pk}'_{\mathsf{San}}\Big)$:
$\pi \leftarrow_\$ \mathsf{Proof}(\mathsf{pp}, m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}'_{\mathsf{San}})$
return $\pi$
```

Figure 10: Privacy (upper-left) and strong privacy (left)

Also this property can be formalized via a security experiment: First, a signer and a sanitizer key pair, as well as a secret bit $b \in \{\mathsf{Sig}, \mathsf{San}\}$, are generated and the public keys are given to an adversary $\mathcal{A}$, who has full (but proof-restricted) adaptive oracle access. He is also allowed to query a left-or-right oracle $\mathcal{O}^{\mathsf{LoR}}$, where he can input a message $m \in \mathcal{M}$ and matching sanitizing rights ADM and modifications MOD. Depending on b, $\mathcal{O}^{\mathsf{LoR}}$ then either signs m (making ADM admissible) and sanitizes the result according to MOD, or first modifies m as specified by MOD and then signs the result. $\mathcal{A}$'s goal is to decide whether he sees a freshly computed signature or a sanitized one, that is to determine the bit b. From here we immediately see why the adversary must be barred from querying the proof oracle on message-signature pairs obtained from $\mathcal{O}^{\mathsf{LoR}}$: This would constitute a trivial attack. SSS is transparent if no efficient adversary as above succeeds in winning the experiment with probability significantly greater than 1/2. The formal definition of transparency is given below.

Definition C.4 (Transparency) *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme and* $\lambda \in \mathbb{N}$ *be a security parameter. Consider an efficient adversary* $\mathcal{A}$ *which:*

a) Takes as input a tuple of public parameters pp *corresponding to the security parameter* λ*, a signer public key* $\mathsf{pk}_{\mathsf{Sig}}$*, and a sanitizer public key* $\mathsf{pk}_{\mathsf{San}}$*;*

b) Has access to a signing oracle $\mathcal{O}^{\mathsf{Sign}}$*, a sanitizing oracle* $\mathcal{O}^{\mathsf{Sanit}}$*, a restricted proof oracle* $\mathcal{O}^{\mathsf{Proof}}$*, and a left-or-right oracle* $\mathcal{O}^{\mathsf{LoR}}$*;*

c) Returns a bit $b^* \in \{\mathsf{Sig}, \mathsf{San}\}$*.*

For every such adversary $\mathcal{A}$, let $\mathbf{Exp}^{\mathsf{Transp}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$ be the experiment defined in Figure 11. We say that SSS *is* transparent *if, for every efficient adversary $\mathcal{A}$ as above, $\mathbf{Adv}^{\mathsf{Transp}}_{\mathcal{A},\mathsf{SSS}}(\lambda) = \mathbb{P}\Big[\mathbf{Exp}^{\mathsf{Transp}}_{\mathcal{A},\mathsf{SSS}}(\lambda) = 1\Big] - 1/2 = \mathsf{negl}(\lambda)$, where the probability is taken over the random coins used by $\mathcal{A}$, as well as the random coins used in the experiment.*

```
$\mathbf{Exp}^{\mathsf{Transp}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:
$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
$(\mathsf{pk_{Sig}}, \mathsf{sk_{Sig}}) \leftarrow_\$ \mathsf{KGen_{Sig}}(\mathsf{pp})$
$(\mathsf{pk_{San}}, \mathsf{sk_{San}}) \leftarrow_\$ \mathsf{KGen_{San}}(\mathsf{pp})$
$b \leftarrow_\$ \{\mathsf{Sig}, \mathsf{San}\}$
$B \leftarrow \emptyset$
$b^* \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}},\mathcal{O}^{\mathsf{Sanit}},\mathcal{O}^{\mathsf{Proof}},\mathcal{O}^{\mathsf{LoR}}}(\mathsf{pp}, \mathsf{pk_{Sig}}, \mathsf{pk_{San}})$
if $b = b^*$ then
    return 1
return 0

$\mathcal{O}^{\mathsf{Sign}}(m, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM})$:
$\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk_{Sig}}, \mathsf{pk_{Sig}}, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM})$
return $\sigma$

$\mathcal{O}^{\mathsf{Sanit}}\Big(m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \mathrm{MOD}\Big)$:
$\sigma' \leftarrow_\$ \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk_{San}}, \mathsf{pk}'_{\mathsf{Sig}}, \mathsf{pk_{San}}, \mathrm{MOD})$
return $\sigma'$

$\mathcal{O}^{\mathsf{Proof}}\Big(m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, \mathsf{pk}'_{\mathsf{San}}\Big)$:
if $\mathsf{pk}'_{\mathsf{San}} = \mathsf{pk_{San}} \ \wedge\ (m, \sigma) \in B$ then
    return $\bot$
$\pi \leftarrow_\$ \mathsf{Proof}(\mathsf{pp}, m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, \mathsf{sk_{Sig}}, \mathsf{pk}'_{\mathsf{San}})$
return $\pi$

$\mathcal{O}^{\mathsf{LoR}}(m, \mathrm{ADM}, \mathrm{MOD})$:
if $\mathrm{ADM}(m) = \bot \ \vee\ \mathrm{MOD}(\mathrm{ADM}) = \bot$ then
    return $\bot$
$m' \leftarrow \mathrm{MOD}(m)$
$\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk_{Sig}}, \mathsf{pk_{Sig}}, \mathsf{pk_{San}}, \mathrm{ADM})$
$\sigma' \leftarrow_\$ \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk_{San}}, \mathsf{pk_{Sig}}, \mathsf{pk_{San}}, \mathrm{MOD})$
if $b = \mathsf{Sig}$ then
    $\sigma' \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m', \mathsf{sk_{Sig}}, \mathsf{pk_{Sig}}, \mathsf{pk_{San}}, \mathrm{ADM})$
$B \leftarrow B \cup \{(m', \sigma')\}$
return $\sigma'$
```

Figure 11: Transparency

(Strong) Signer-Accountability. The main motivation behind sanitizable signature schemes is to allow a semi-trusted party, the sanitizer, to alter signed data and then to update the signature. Under these circumstances, it is very important to always be able to determine who is reponsible for a given message-signature pair—the signer or the sanitizer. *Signer-accountability* requires that a malicious signer cannot deny that a given message-signature pair originated from himself, if it was indeed him who put the signature.
In the security experiment defining signer-accountability, a sanitizer key pair is first generated and the public key is given to the adversary $\mathcal{A}$, who has full adaptive access to the sanitizing oracle. $\mathcal{A}$'s goal is to impersonate a legitimate signer (here identified by the public key $\mathsf{pk}^*_{\mathsf{Sig}}$), and to come up with a valid message-signature pair (m^*, σ^*) and a proof π^* which tricks Judge into blaming the sanitizer for (m^*, σ^*), even though the signer is actually responsible for it. SSS is signer-accountable if any adversary as above succeeds in doing so only with negligible probability.
Signer-accountability can be further strengthened to also account for the signatures, in very much the same way as strong unforgeability arises from (regular) unforgeability. The formal definition of (strong) signer-accountability is given below.

Definition C.5 ((Strong) Signer-Accountability) *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme and* $\lambda \in \mathbb{N}$ *be a security parameter. Consider an efficient adversary* $\mathcal{A}$ *which:*

a) Takes as input a tuple of public parameters pp *corresponding to the security parameter* λ *and a sanitizer public key* $\mathsf{pk}_{\mathsf{San}}$*;*

b) Has access to a sanitizing oracle $\mathcal{O}^{\mathsf{Sanit}}$ *(resp.* $\tilde{\mathcal{O}}^{\mathsf{Sanit}}$*);*

c) Returns a tuple $\left(m^*, \sigma^*, \pi^*, \mathsf{pk}^*_{\mathsf{Sig}}\right) \in \mathcal{M} \times \mathcal{S} \times \{0,1\}^* \times \mathcal{K}_{\mathsf{Sig,pk}}$.

For every such adversary $\mathcal{A}$*, let* $\mathbf{Exp}^T_{\mathcal{A},\mathsf{SSS}}(\lambda)$ *with* $T = \mathsf{SigAcc}$ *(resp.* $T = \mathsf{SSigAcc}$*) be the experiment defined in the upper-left corner (resp. in the lower-left corner) of Figure 12. We say that* SSS *is* signer-accountable *(resp.* strongly signer-accountable*) if, for every efficient adversary* $\mathcal{A}$ *as above,* $\mathbf{Adv}^T_{\mathcal{A},\mathsf{SSS}}(\lambda) = \mathbb{P}\left[\mathbf{Exp}^T_{\mathcal{A},\mathsf{SSS}}(\lambda) = 1\right] = \mathsf{negl}(\lambda)$*, where the probability is taken over the random coins used by* $\mathcal{A}$*, as well as the random coins used in the experiment.*

```
$\mathbf{Exp}^{\mathsf{SigAcc}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:
$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$
$B \leftarrow \emptyset$
$\left(m^*, \sigma^*, \pi^*, \mathsf{pk}^*_{\mathsf{Sig}}\right) \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{Sanit}}}(\mathsf{pp}, \mathsf{pk}_{\mathsf{San}})$
$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}^*_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
$d' \leftarrow \mathsf{Judge}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}^*_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \pi^*)$
if $d = \top \wedge d' = \mathsf{San} \wedge (\mathsf{pk}^*_{\mathsf{Sig}}, m^*) \notin B$ then
    return 1
return 0
```

```
$\mathbf{Exp}^{\mathsf{SSigAcc}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:
$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$
$B \leftarrow \emptyset$
$\left(m^*, \sigma^*, \pi^*, \mathsf{pk}^*_{\mathsf{Sig}}\right) \leftarrow_\$ \mathcal{A}^{\tilde{\mathcal{O}}^{\mathsf{Sanit}}}(\mathsf{pp}, \mathsf{pk}_{\mathsf{San}})$
$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}^*_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
$d' \leftarrow \mathsf{Judge}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}^*_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \pi^*)$
if $d = \top \wedge d' = \mathsf{San} \wedge (\mathsf{pk}^*_{\mathsf{Sig}}, m^*, \sigma^*) \notin B$ then
    return 1
return 0
```

```
$\mathcal{O}^{\mathsf{Sanit}}\left(m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \mathrm{MOD}\right)$:
$m' \leftarrow \mathrm{MOD}(m)$
$\sigma' \leftarrow_\$ \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}'_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD})$
$B \leftarrow B \cup \{(\mathsf{pk}'_{\mathsf{Sig}}, m')\}$
return $\sigma'$
```

```
$\tilde{\mathcal{O}}^{\mathsf{Sanit}}\left(m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \mathrm{MOD}\right)$:
$m' \leftarrow \mathrm{MOD}(m)$
$\sigma' \leftarrow_\$ \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}'_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD})$
$B \leftarrow B \cup \{(\mathsf{pk}'_{\mathsf{Sig}}, m', \sigma')\}$
return $\sigma'$
```

Figure 12: Signer-accountability (left) and strong signer-accountability (right)

(Strong) Sanitizer-Accountability. The counterpart of signer-accountability, with all the roles reversed, is called *sanitizer-accountability.* In the corresponding security experiment, a signer key pair is first generated and the public key is given to the adversary $\mathcal{A}$, who has full adaptive oracle access. $\mathcal{A}$'s goal is to impersonate a legitimate sanitizer (here identified by the public key $\mathsf{pk}^*_{\mathsf{San}}$), and to come up with a valid message-signature pair (m^*, σ^*) such that, when the signer generates a proof π, Judge will blame the signer for (m^*, σ^*), even though the sanitizer is actually responsible for the signature. Of course one should exclude the trival attack where the adversary simply returns a message-signature pair he got from the signing oracle. SSS is sanitizer-accountable if any adversary as above succeeds in winning the experiment only with negligible probability.
Again, sanitizer-accountability can be further strengthened to also account for the signatures. The formal definition of (strong) sanitizer-accountability is given below.

Definition C.6 ((Strong) Sanitizer-Accountability) *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme and* $\lambda \in \mathbb{N}$ *be a security parameter. Consider an efficient adversary* $\mathcal{A}$ *which:*

a) Takes as input a tuple of public parameters pp *corresponding to the security parameter* λ *and a signer public key* $\mathsf{pk}_{\mathsf{Sig}}$*;*

b) Has access to a signing oracle $\mathcal{O}^{\mathsf{Sign}}$ *and a proof oracle* $\mathcal{O}^{\mathsf{Proof}}$*;*

c) Returns a tuple $(m^*, \sigma^*, \mathsf{pk}^*_{\mathsf{San}}) \in \mathcal{M} \times \mathcal{S} \times \mathcal{K}_{\mathsf{San,pk}}$*.*

For every such adversary $\mathcal{A}$*, let* $\mathbf{Exp}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda)$ *with* $T = \mathsf{SanAcc}$ *(resp.* $T = \mathsf{SSanAcc}$*) be the experiment defined on the left-hand side (resp. right-hand side) of Figure 13. We say that* SSS *is* sanitizer-accountable *(resp.* strongly sanitizer-accountable*) if, for every efficient adversary* $\mathcal{A}$ *as above,* $\mathbf{Adv}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda) = \mathbb{P}\left[\mathbf{Exp}^{T}_{\mathcal{A},\mathsf{SSS}}(\lambda) = 1\right] = \mathsf{negl}(\lambda)$*, where the probability is taken over the random coins used by* $\mathcal{A}$*, as well as the random coins used in the experiment.*

```
$\mathbf{Exp}^{\mathsf{SanAcc}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:
$\mathsf{pp} \leftarrow_{\$} \mathsf{PGen}(1^{\lambda})$
$(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$
$L \leftarrow \emptyset,\ B \leftarrow \emptyset$
$(m^*, \sigma^*, \mathsf{pk}^*_{\mathsf{San}}) \leftarrow_{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Proof}}}(\mathsf{pp}, \mathsf{pk}_{\mathsf{Sig}})$
$\pi \leftarrow_{\$} \mathsf{Proof}(\mathsf{pp}, m^*, \sigma^*, L, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}^*_{\mathsf{San}})$
$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}^*_{\mathsf{San}})$
$d' \leftarrow \mathsf{Judge}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}^*_{\mathsf{San}}, \pi)$
if $d = \top \ \wedge\ d' = \mathsf{Sig} \ \wedge\ (\mathsf{pk}^*_{\mathsf{San}}, m^*) \notin B$ then
    return 1
return 0

$\mathcal{O}^{\mathsf{Sign}}(m, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM})$:
$\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM})$
$L \leftarrow L \cup \{(m, \sigma)\}$
$B \leftarrow B \cup \{(\mathsf{pk}'_{\mathsf{San}}, m)\}$
return $\sigma$
```

```
$\mathbf{Exp}^{\mathsf{SSanAcc}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:
$\mathsf{pp} \leftarrow_{\$} \mathsf{PGen}(1^{\lambda})$
$(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_{\$} \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$
$L \leftarrow \emptyset,\ B \leftarrow \emptyset$
$(m^*, \sigma^*, \mathsf{pk}^*_{\mathsf{San}}) \leftarrow_{\$} \mathcal{A}^{\tilde{\mathcal{O}}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Proof}}}(\mathsf{pp}, \mathsf{pk}_{\mathsf{Sig}})$
$\pi \leftarrow_{\$} \mathsf{Proof}(\mathsf{pp}, m^*, \sigma^*, L, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}^*_{\mathsf{San}})$
$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}^*_{\mathsf{San}})$
$d' \leftarrow \mathsf{Judge}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}^*_{\mathsf{San}}, \pi)$
if $d = \top \ \wedge\ d' = \mathsf{Sig} \ \wedge\ (\mathsf{pk}^*_{\mathsf{San}}, m^*, \sigma^*) \notin B$ then
    return 1
return 0

$\tilde{\mathcal{O}}^{\mathsf{Sign}}(m, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM})$:
$\sigma \leftarrow_{\$} \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM})$
$L \leftarrow L \cup \{(m, \sigma)\}$
$B \leftarrow B \cup \{(\mathsf{pk}'_{\mathsf{San}}, m, \sigma)\}$
return $\sigma$
```

```
$\mathcal{O}^{\mathsf{Proof}}\left(m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^{k}, \mathsf{pk}'_{\mathsf{San}}\right)$:
$\pi \leftarrow_{\$} \mathsf{Proof}(\mathsf{pp}, m, \sigma, \{(m_i, \sigma_i)\}_{i=1}^{k}, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}'_{\mathsf{San}})$
return $\pi$
```

Figure 13: Sanitizer-accountability (left) and strong sanitizer-accountability (right)

Non-Interactive Public Accountability. In the original model of sanitizable signatures (see Brzuska *et al.* [BFF+09]), the party responsible for a given message-signature pair can in general be identified only with the aid of a proof $\pi \in \{0,1\}^*$, which can be generated if the signer secret key $\mathsf{sk}_{\mathsf{Sig}}$ is known. In many cases, this contradics legal and application requirements (see e.g. Brzuska *et al.* [BPS12, BPS13] for further discussions). *Non-interactive public accountability* addresses these shortcomings: It requires that the party responsible for a given message-signature pair can be determined without any additional information from either the signer or the sanitizer.

In the security experiment defining non-interactive public accountability, first a signer and a sanitizer key pair are generated and the public keys are given to the adversary $\mathcal{A}$, who has full adaptive access to a signing and a sanitizing oracle. $\mathcal{A}$'s goal is to impersonate either a legitiate signer or a legitimate sanitizer (here identified by the public key pk^*), and to come up with a valid message-signature pair (m^*, σ^*) such that Judge, without any external help (here modelled by the trivial proof $\pi = \bot$), attributes it to the wrong party. SSS is non-interactive publicly accountable if any adversary as above succeeds in doing so only with negligible probability. The formal definition of non-interactive public accountability is given below.

Definition C.7 (Non-Interactive Public Accountability) *Let* $\mathsf{SSS} := (\mathsf{PGen}, \mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a sanitizable signature scheme and* $\lambda \in \mathbb{N}$ *be a security parameter. Consider an efficient adversary* $\mathcal{A}$ *which:*

a) Takes as input a tuple of public parameters pp *corresponding to the security parameter* λ*, a signer public key* $\mathsf{pk}_{\mathsf{Sig}}$*, and a sanitizer public key* $\mathsf{pk}_{\mathsf{San}}$*;*

b) Has access to a signing oracle $\mathcal{O}^{\mathsf{Sign}}$ *and a sanitizing oracle* $\mathcal{O}^{\mathsf{Sanit}}$*;*

c) Returns a tuple $(m^*, \sigma^*, \mathsf{pk}^*) \in \mathcal{M} \times \mathcal{S} \times (\mathcal{K}_{\mathsf{Sig,pk}} \cup \mathcal{K}_{\mathsf{San,pk}})$*.*

For every such adversary $\mathcal{A}$*, let* $\mathbf{Exp}^{\mathsf{NIPA}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$ *be the experiment defined in Figure 14. We say that* SSS *is* non-interactive publicly accountable *if, for every efficient adversary* $\mathcal{A}$ *as above,* $\mathbf{Adv}^{\mathsf{NIPA}}_{\mathcal{A},\mathsf{SSS}}(\lambda) = \mathbb{P}\Big[\mathbf{Exp}^{\mathsf{NIPA}}_{\mathcal{A},\mathsf{SSS}}(\lambda) = 1\Big] = \mathsf{negl}(\lambda)$*, where the probability is taken over the random coins used by* $\mathcal{A}$*, as well as the random coins used in the experiment.*

```
$\mathbf{Exp}^{\mathsf{NIPA}}_{\mathcal{A},\mathsf{SSS}}(\lambda)$:
$\mathsf{pp} \leftarrow_\$ \mathsf{PGen}(1^\lambda)$
$(\mathsf{pk}_{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{Sig}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{Sig}}(\mathsf{pp})$
$(\mathsf{pk}_{\mathsf{San}}, \mathsf{sk}_{\mathsf{San}}) \leftarrow_\$ \mathsf{KGen}_{\mathsf{San}}(\mathsf{pp})$
$B_{\mathsf{Sign}} \leftarrow \emptyset,\ B_{\mathsf{Sanit}} \leftarrow \emptyset$
$(m^*, \sigma^*, \mathsf{pk}^*) \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sanit}}}(\mathsf{pp}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}})$
$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}^*)$
$d' \leftarrow \mathsf{Judge}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}^*, \bot)$
if $d = \top \wedge d' = \mathsf{Sig} \wedge (m^*, \sigma^*, \mathsf{pk}^*) \notin B_{\mathsf{Sign}}$ then
    return 1
$d \leftarrow \mathsf{Verify}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk}_{\mathsf{San}})$
$d' \leftarrow \mathsf{Judge}(\mathsf{pp}, m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk}_{\mathsf{San}}, \bot)$
if $d = \top \wedge d' = \mathsf{San} \wedge (m^*, \sigma^*, \mathsf{pk}^*) \notin B_{\mathsf{Sanit}}$ then
    return 1
return 0

$\mathcal{O}^{\mathsf{Sign}}(m, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM})$:
$\sigma \leftarrow_\$ \mathsf{Sign}(\mathsf{pp}, m, \mathsf{sk}_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{Sig}}, \mathsf{pk}'_{\mathsf{San}}, \mathrm{ADM})$
$B_{\mathsf{Sign}} \leftarrow B_{\mathsf{Sign}} \cup \{(m, \sigma, \mathsf{pk}'_{\mathsf{San}})\}$
return $\sigma$

$\mathcal{O}^{\mathsf{Sanit}}\Big(m, \sigma, \mathsf{pk}'_{\mathsf{Sig}}, \mathrm{MOD}\Big)$:
$m' \leftarrow \mathrm{MOD}(m)$
$\sigma' \leftarrow_\$ \mathsf{Sanit}(\mathsf{pp}, m, \sigma, \mathsf{sk}_{\mathsf{San}}, \mathsf{pk}'_{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{San}}, \mathrm{MOD})$
$B_{\mathsf{Sanit}} \leftarrow B_{\mathsf{Sanit}} \cup \{(m', \sigma', \mathsf{pk}'_{\mathsf{Sig}})\}$
return $\sigma'$
```

Figure 14: Non-interactive public accountability