A preliminary version appears in Public-Key Cryptography (PKC) 2009, Lecture Notes in Computer Science, Springer-Verlag, 2009.

Security of Sanitizable Signatures Revisited

Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann,
Marcus Page, Jakob Schelbert, Dominique Schröder, Florian Volk

Darmstadt University of Technology, Germany
www.minicrypt.de

Abstract. Sanitizable signature schemes, as defined by Ateniese et al. (ESORICS 2005), allow a signer to partly delegate signing rights to another party, called the sanitizer. That is, the sanitizer is able to modify a predetermined part of the original message such that the integrity and authenticity of the unchanged part is still verifiable. Ateniese et al. identify five security requirements for such schemes (unforgeability, immutability, privacy, transparency and accountability) but do not provide formal specifications for these properties. They also present a scheme that is supposed to satisfy these requirements.

Here we revisit the security requirements for sanitizable signatures and, for the first time, present a comprehensive formal treatment. Besides a full characterization of the requirements we also investigate the relationship of the properties, showing for example that unforgeability follows from accountability. We then provide a full security proof for a modification of the original scheme according to our model.

1 Introduction

Sanitizable signature schemes, introduced by Ateniese et al. [ACdMT05] and, in a slightly different vein, by Steinfeld et al. [SBZ01] and Miyazaki et al. [MSI+03], allow a signer to delegate signature rights in a controlled way. Namely, the signer can determine parts of the message which a designated party, the sanitizer, can later modify but such that the authenticity and integrity of the remaining parts is still guaranteed. In particular, even the sanitizer should not be able to change inadmissible parts of the message and produce a valid signature for such illegitimate transformations.

A straightforward application of sanitizable signatures are medical data which should be published in an anonymized but authentic form. Suppose for example that for infectious disease surveillance a hospital is obliged to report excerpts of their patients medical data like dates of birth, genders etc. to an authority. Yet other parts of these data can and should be anonymized, e.g., pseudonyms replacing the patients names or deleting psychiatric information.

Ideally, the administrative department of the hospital assembles the requested information from their records, holding the medical data signed by different health professionals, and sanitizes them without further interaction with their personnel. At the same time the authenticity and integrity of the dedicated data should be preserved. Then, clearly, sanitizable signatures in which the hospital acts as a sanitizer provide a solution. Ateniese et al. [ACdMT05] provide further applications of sanitizable signature schemes, including multicast, data base outsourcing and secure routing.

Security Requirements. As discussed in [ACdMT05] meaningful sanitizable signatures come with the usual unforgeability requirement of regular signature schemes:

Unforgeability. It should be infeasible for an outsider (i.e., neither the signer nor the sanitizer) to forge signatures in the name of the signer or the sanitizer.

But the introduction of the sanitizing party and its relationship to the signer entail further desirable security properties. These are:

Immutability. The sanitizer should not be able to produce valid signatures for messages where it has changed other than the designated parts (this can be thought of as an insider attack).

Privacy. Sanitized messages and their signatures should not reveal the original data (i.e., the parts which have been sanitized).

Transparency. It should be infeasible to decide whether a message has been sanitized or not. This may be desirable in applications where one should not be able to discriminate against messages produced by the sanitizer.

Accountability. A party (the signer or the sanitizer) should not be held responsible for messages originating from the other party.

While unforgeability can be formalized straightforwardly from the basic case for regular signatures, as it is done in [ACdMT05], Ateniese et al. remain rather vague when it comes to the other security requirements. Instead, they introduce technical conditions for the sanitizable signature scheme, aiming to achieve the requirements above. Besides unforgeability these are indistinguishability —roughly saying that signatures generated by the signer are computationally independent of the messages— and the property of identical distributions, saying that the signatures produced by the signer and the sanitizer have identical distributions. This approach is arguable in several ways.

First, without having a formal definition of the security requirements above it is hard to tell if a signature scheme with the technical conditions really achieves the desired goals; as always in cryptography, without a robust security model underneath it is impossible to make precise statements about the hardness of attacks. Secondly, having a more abstract view on the desirable security requirements (instead of the scheme's conditions) facilitates the understanding of their relationships among each other and with other cryptographic primitives. Finally, trying to achieve the security requirements via technical properties seems to be exceedingly restrictive and may exclude otherwise viable solutions.

Our Results. In this paper we revisit the aforementioned security requirements and formalize them according to common game-based approaches. As part of this, we simplify the unforgeability experiment from [ACdMT05]. We also make several refinements for accountability. First, we augment the model by new algorithms Proof and Judge where Proof allows to provide evidence to Judge that a message has been sanitized. Then we distinguish between *sanitizer-* and *signer-accountability*, saying that a malicious sanitizer resp. signer cannot falsely accuse the other party. The original approach in [ACdMT05] only seems to discuss our notion of sanitizer-accountability.

Concerning the relationship of the now-defined security requirements we obtain some useful and also some unexpected results: First, we prove that transparency implies privacy, i.e., any transparent sanitizable signature scheme is also private and for such schemes there is no need to look at the privacy property separately. Secondly, we show that the two accountability types together imply unforgeability, which is in contrast to the position of Ateniese et al. [ACdMT05] who argue that unforgeability implies accountability. Having a clean model tells us that it is the other way around, and that accountability needs to be considered.

As for the other security properties, immutability, transparency, sanitizer- and signer-accountability we show that each property is independent of the other ones. That is, for each property we present a sanitizable scheme which satisfies all the other requirements except for the one in question. Technically we assume that there are schemes having all properties and then modify the scheme to annihilate the one property. Finally, we show that unforgeability does not follow from sanitizer- or signer-accountability alone (but only if both versions of accountability hold simultaneously). This gives us a complete characterization of the relationship of the notions.

We also revisit the sanitizable signature scheme presented in [ACdMT05] in light of our formal definitions. We show that a modification of their scheme indeed meets our requirements for immutability, transparency,

sanitizer-accountability and signer-accountability. This already implies, via our relationship results, that the scheme is also unforgeable and private and thus a secure sanitizable scheme.

Related Work. As mentioned before, Miyazaki et al. [MSI+03] also use the notion of sanitizable signature schemes, but refer to a slightly different approach. According to their notion only deletions of message parts are considered (instead of modifications) and, secondly, the sanitizer is usually not bound to change designated parts of the message but can decide which portions should be deleted. The basic security properties of such sanitizable signature schemes are unforgeability and privacy (following the terminology above). Independently, several similar proposals like content extraction signatures [SBZ01] and redactable signatures [JMSW02] have been made.

The two approaches for sanitizable signatures and their solutions resemble each other, making the distinction somewhat obscure. This is especially true since further properties have been added to the models in subsequent works, like the requirement that the sanitizer's identity remains hidden [MHI08] in the sanitizable signature model of [MSI+03], resembling the above notion of transparency. Nonetheless, one can divide the literature about sanitizable signatures roughly into the works following the approach by Ateniese et al., e.g., [KL06, CLM08], and the works based on the approach by Miyazaki et al., including [IKTY05, MIM+05, IKO+07, MHI08, HHH+08].

We adhere to the notion of sanitizable signature of Ateniese et al. [ACdMT05], covering message modifications and security requirements like accountability. Some improvements concerning the scheme's efficiency have been made [KL06] and some extensions concerning multiple, a-posteriori determined censors have been suggested [CLM08]. None of these proposals goes beyond the original approach to model the security properties formally, though. We note that some of the previous works in the vein of Miyazaki et al. [MSI+03] come with security models, especially for privacy and unforgeability [SBZ01, MHI08, SIT06]. Yet, they often provide limited security guarantees, like privacy requirements holding for a single message-signature pair only. In contrast our models allow more sophisticated attacks where for instance privacy should still hold for multiple message-signature pairs and even if the attacker can ask for further signatures.

Independently of our work, Yuen et al. [YSLM08] also revisit the security of sanitizable signatures, but focus on new constructions.

2 Preliminaries

In this section we define sanitizable signatures. Like a regular signature scheme a sanitizable signature scheme allows to sign messages under the secret signer key sk_{sig}, generated together with the public verification key pk_{sig}. The signing process itself includes a public key pk_{san} of a designated sanitizer and a description ADM of division into blocks and admissible blocks which the sanitizer is allowed to change with the help of its secret key sk_{san}. Any such modification takes the original message and signature and some modification information MOD and produces a signature σ' for the modified message m'.

In the sequel we assume for simplicity that the description ADM of admissible blocks defines the block length $t \in \mathbb{N}$ and contains a set of block numbers from $\mathbb{N}$ which can be changed, and that all messages are aligned to block length (say, by standard padding techniques). The modification information MOD is then a list of pairs $(j, m'[j])$ consisting of a block number j and the new content $m'[j]$ for this block. We say that MOD *matches* ADM if all the block numbers in MOD are admissible according to ADM and the length of the blocks in MOD equals the value in ADM. The case of a more general transformation, where the modifications are modeled as arbitrary algorithms, is straightforward and discussed in Appendix B.

In addition, to settle disputes about the origin of a message-signature pair, an algorithm Proof enables the signer to produce a proof π that a signature has been created by the sanitizer. The proof π is generated from a set of previously signed messages. A Judge algorithm then uses the proof π to decide if a *valid* message-signature pair (m, σ) has been created by the signer or the sanitizer (the lack of such a proof is interpreted as a signer origin). We note that Judge is usually only called for valid pairs (m, σ); for invalid pairs settling the dispute is beyond the scheme's scope.

Definition 2.1 (Sanitizable Signature Scheme) *A sanitizable signature scheme* SanSig *consists of seven efficient algorithms* ($\mathsf{KGen}_{sig}, \mathsf{KGen}_{san}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge}$) *such that:*

KEY GENERATION. *There are two key generation algorithms, one for the signer and one for the sanitizer. Both create a pair of keys, a private key and the corresponding public key:*

$$(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^n), \qquad (pk_{san}, sk_{san}) \leftarrow \mathsf{KGen}_{san}(1^n)$$

SIGNING. *The* Sign *algorithm takes as input a message $m \in \{0,1\}^*$, the secret key sk_{sig} of the signer, the public key pk_{san} of the sanitizer as well as a description* ADM $\in \mathbb{N} \times 2^{\mathbb{N}}$ *of the block length t and admissibly modifiable message blocks from $\{0,1\}^t$. It outputs a signature (or $\perp$, indicating an error):*

$$\sigma \leftarrow \mathsf{Sign}(m, sk_{sig}, pk_{san}, \mathrm{ADM}).$$

We assume that ADM *is recoverable from any signature $\sigma \neq \perp$.*

SANITIZING. *Algorithm* Sanit *takes a message $m \in \{0,1\}^*$, a signature σ, the public key pk_{sig} of the signer and the secret key sk_{san} of the sanitizer. It modifies the message m according to the modification instruction* MOD $\subseteq \mathbb{N} \times \{0,1\}^t$ *(where t is the block length described in* ADM*) and determines a new signature σ' for the modified message m'. Then* Sanit *outputs m' and σ' (or possibly $\perp$ in case of an error).*

$$(m', \sigma') \leftarrow \mathsf{Sanit}(m, \mathrm{MOD}, \sigma, pk_{sig}, sk_{san})$$

VERIFICATION. *The* Verify *algorithm outputs a bit $d \in \{\mathtt{true}, \mathtt{false}\}$ verifying the correctness of a signature σ for a message m with respect to the public keys pk_{sig} and pk_{san}.*

$$d \leftarrow \mathsf{Verify}(m, \sigma, pk_{sig}, pk_{san})$$

PROOF. *The* Proof *algorithm takes as input the secret signing key sk_{sig}, a message m and a signature σ as well a set of (polynomially many) additional message-signature pairs $(m_i, \sigma_i)_{i=1,2,\ldots,q}$ and the public key pk_{san}. It outputs a string $\pi \in \{0,1\}^*$:*

$$\pi \leftarrow \mathsf{Proof}(sk_{sig}, m, \sigma, (m_1, \sigma_1), \ldots, (m_q, \sigma_q), pk_{san})$$

JUDGE. *Algorithm* Judge *takes as input a message m and a valid signature σ, the public keys of the parties and a proof π. It outputs a decision $d \in \{\mathtt{Sig}, \mathtt{San}\}$ indicating whether the message-signature pair has been created by the signer or the sanitizer:*

$$d \leftarrow \mathsf{Judge}(m, \sigma, pk_{sig}, pk_{san}, \pi)$$

For a sanitizable signature scheme the usual correctness properties should hold, saying that genuinely signed or sanitized messages are accepted and that a genuinely created proof by the signer leads the judge to decide in favor of the signer.

SIGNING CORRECTNESS. For any security parameter $n \in \mathbb{N}$, any key pair $(sk_{\mathrm{sig}}, pk_{\mathrm{sig}}) \leftarrow \mathsf{KGen}_{\mathrm{sig}}(1^n)$, any key pair $(sk_{\mathrm{san}}, pk_{\mathrm{san}}) \leftarrow \mathsf{KGen}_{\mathrm{san}}(1^n)$, any message $m \in \{0,1\}^*$, any ADM $\in \mathbb{N} \times 2^{\mathbb{N}}$ and any $\sigma \leftarrow \mathsf{Sign}(m, sk_{\mathrm{sig}}, pk_{\mathrm{san}}, \mathrm{ADM})$ we have

$$\mathsf{Verify}(m, \sigma, pk_{\mathrm{sig}}, pk_{\mathrm{san}}) = \mathtt{true}.$$

SANITIZING CORRECTNESS. For any security parameter $n \in \mathbb{N}$, any key pair $(sk_{\mathrm{sig}}, pk_{\mathrm{sig}}) \leftarrow \mathsf{KGen}_{\mathrm{sig}}(1^n)$, any key pair $(sk_{\mathrm{san}}, pk_{\mathrm{san}}) \leftarrow \mathsf{KGen}_{\mathrm{san}}(1^n)$, any message $m \in \{0,1\}^*$, any σ with $\mathsf{Verify}(m, \sigma, pk_{\mathrm{sig}}, pk_{\mathrm{san}}) = \mathtt{true}$, any MOD $\subseteq \mathbb{N} \times \{0,1\}^t$ matching ADM from σ, and any pair $(m', \sigma') \leftarrow \mathsf{Sanit}(m, \mathrm{MOD}, \sigma, pk_{\mathrm{sig}}, sk_{\mathrm{san}})$ we require

$$\mathsf{Verify}(m', \sigma', pk_{\mathrm{sig}}, pk_{\mathrm{san}}) = \mathtt{true}.$$

PROOF CORRECTNESS. For any security parameter $n \in \mathbb{N}$, any key pair $(sk_{\text{sig}}, pk_{\text{sig}}) \leftarrow \mathsf{KGen}_{\text{sig}}(1^n)$, any key pair $(sk_{\text{san}}, pk_{\text{san}}) \leftarrow \mathsf{KGen}_{\text{san}}(1^n)$, any message $m \in \{0,1\}^*$, any signature σ, any MOD matching ADM from σ, any $(m', \sigma') \leftarrow \mathsf{Sanit}(m, \text{MOD}, \sigma, pk_{\text{sig}}, sk_{\text{san}})$ with $\mathsf{Verify}(m', \sigma', pk_{\text{sig}}, pk_{\text{san}}) = \texttt{true}$, and any (polynomially many) $m_1, \ldots, m_q$ and $\text{ADM}_1, \ldots, \text{ADM}_q$ with $\sigma_i \leftarrow \mathsf{Sign}(m_i, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM}_i)$ and $(m, \sigma) = (m_i, \sigma_i)$ for some i, any $\pi \leftarrow \mathsf{Proof}(sk_{\text{sig}}, m', \sigma', m_1, \sigma_1, \ldots, m_q, \sigma_q, pk_{\text{san}})$ we require:

$$\mathsf{Judge}(m', \sigma', pk_{\text{sig}}, pk_{\text{san}}, \pi) = \mathtt{San}.$$

3 Security Requirements

According to Ateniese et al. [ACdMT05] there are several security requirements that a secure sanitizable signature needs to satisfy. Informally, these are:

UNFORGEABILITY. No outsider should be able to forge the signer's or the censor's signature. This is analogously to the standard security requirement for signatures.

IMMUTABILITY. The censor is allowed to modify predefined, admissible parts of a message, but he should not be able to modify other parts of the message. For example, a sanitizer who is in charge of blackening names in medical documents should not be able to modify the actual medical data.

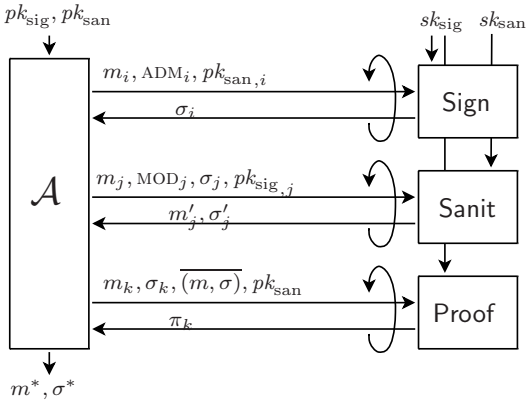
PRIVACY. Nobody should be able to restore sanitized parts of a message. For example, if we have pseudonyms in medical documents then, of course, the original names should not be recoverable.

TRANSPARENCY. The idea of sanitizable signatures is that, within well-defined limits, the sanitizer inherits the signing authority. Sometimes knowledge of this fact makes the sanitized data less valuable, e.g., an original business plan coming from the CEO is a more desirable target for a spy than the sanitized plan from the administration office. Transparency now says that no one except for the signer and the sanitizer should be able to distinguish signatures from the signer and the sanitizer.

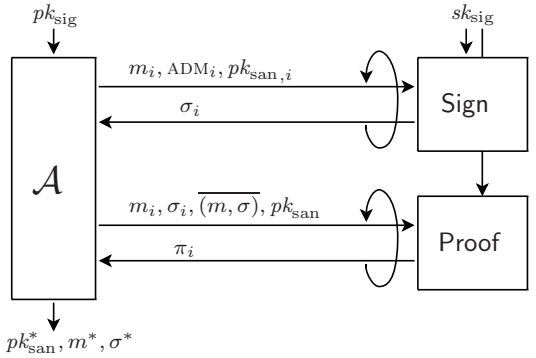
ACCOUNTABILITY. If the signer and the censor have an argument about the origin of a valid message-signature pair (m, σ), then accountability demands that this dispute can be settled correctly by the Judge. As an example consider a public servant acting as a sanitizer, but publishing unauthorized information in the name of the government.

We next define these notions formally. An overview is given in Figure 1. We note that we call a sanitizable scheme *secure* if it is simultaneously immutable, unforgeable, private, transparent, sanitizer-accountable and signer-accountable according to the definitions below.

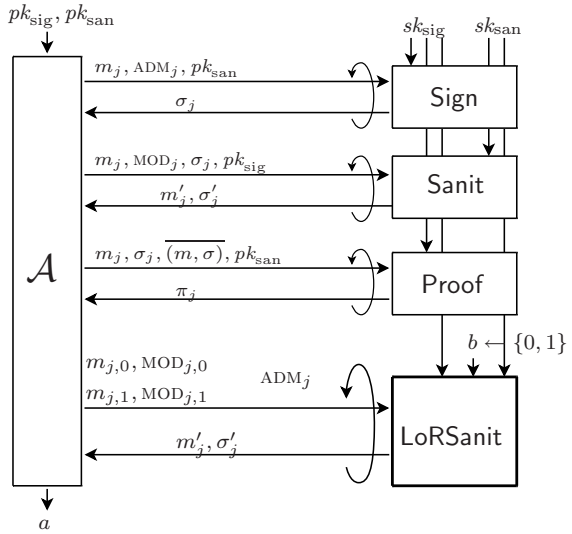
We note that our definitions usually consider three parties, the signer, the sanitizer and the adversary (for some properties the adversary takes over the role of one of the other two parties). In practice, though, one usually has many parties, e.g., a sanitizer assigned to many signers. Our definitions are robust in this regard as we leave much power to the adversary and its queries, say, asking the honest signer to sign a message for a chosen public sanitizer key and thus for different sanitizers. By this, our models can be easily mapped to the case of multiple parties by standard guessing strategies (i.e., trying to predict the "target" signer-sanitizer pair and simulating the other honest parties). As an example we show that our notion of immutability also provides security against the "additional sanitizing attack" of [MIM+05], a typical non-malleability attack for three parties.



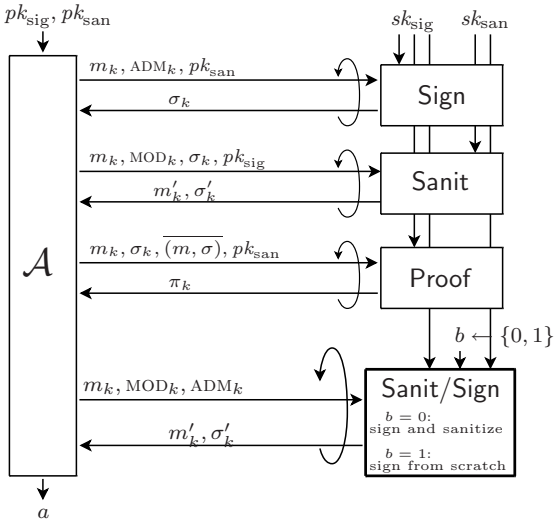
(a) Unforgeability: $\mathcal{A}$'s output has to verify as true and $\forall i = 1, \ldots, q : (pk_{\text{san}}, m^*) \neq (pk_{\text{san},i}, m_i)$ and $\forall j = q+1, \ldots, r : (pk_{\text{sig}}, m^*) \neq (pk_{\text{sig},j}, m'_j)$.



(b) Immutability: $\mathcal{A}$'s output has to verify as true and for all $i = 1, \ldots, q$ it must holds that $pk^*_{\text{san}} \neq pk_{\text{san},i}$ or $m^*[j_i] \neq m_i[j_i]$ for some $j_i \notin \text{ADM}_i$.



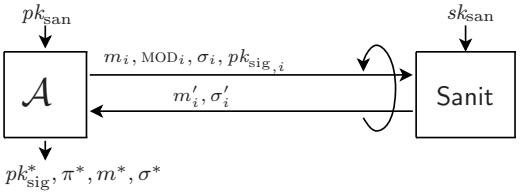
(c) Privacy: $\mathcal{A}$ wins if it outputs $a = b$.



(d) Transparency: $\mathcal{A}$ wins if it outputs $a = b$.

Sanitizer Accountability: The adversary is a malicious sanitizer and has the same input and oracles as in the Immutability-Experiment above. Here $\mathcal{A}$ wins if the corresponding proof to its output $(pk^*_{\text{sig}}, m^*, \sigma^*)$ leads Judge to decide that the signature has been generated by the signer.

(e)



(f) Signer Accountability: $\mathcal{A}$ wins if it outputs a proof π^* for some valid message/signature pair $(pk^*_{\text{sig}}, m^*, \sigma^*)$ which Judge relates to the sanitizer.

Figure 1: Security Requirements of Sanitizable Signatures

3.1 Unforgeability

The unforgeability notion for sanitizable signatures follows the classical notion for regular signature schemes. It says that nobody should be able to compute a tupel (m^*, σ^*) such that $\mathsf{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk_{\text{san}}) = \texttt{true}$ without having the secret keys $sk_{\text{sig}}, sk_{\text{san}}$. This must hold even if one can see additional signatures for other messages. We also give the adversary access to a Proof box (as proofs could potentially leak information about the secret signing key). Yet, except for this secret key the adversary fully determines the other input data, including the message-signature pairs and the public keys. This allows to capture for example scenarios where several sanitizers are assigned to the same signer.

Definition 3.1 (Unforgeability) *A sanitizable signature scheme* SanSig *is* unforgeable *if for any efficient algorithm $\mathcal{A}$ the probability that the following experiment returns 1 is negligible (as a function of n):*

Experiment $\mathsf{Unforgeability}^{\mathsf{SanSig}}_{\mathcal{A}}(n)$
$(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^n)$
$(pk_{san}, sk_{san}) \leftarrow \mathsf{KGen}_{san}(1^n)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{sig}, \cdot, \cdot), \mathsf{Sanit}(\cdot, \cdot, \cdot, \cdot, sk_{san}), \mathsf{Proof}(sk_{sig}, \dots)}(pk_{sig}, pk_{san})$
letting $(m_i, \text{ADM}_i, pk_{san,i})$ *and* σ_i *for* $i = 1, 2, \dots, q$
denote the queries and answers to and from oracle Sign,
and $(m_j, \text{MOD}_j, \sigma_j, pk_{sig,j})$ *and* (m'_j, σ'_j) *for* $j = q+1, \dots, r$
denote the queries and answers to and from oracle Sanit.
return 1 *if*
$\mathsf{Verify}(m^*, \sigma^*, pk_{sig}, pk_{san}) = \texttt{true}$ *and*
for all $i = 1, 2, \dots, q$ *we have* $(pk_{san}, m^*) \neq (pk_{san,i}, m_i)$ *and*
for all $j = q+1, \dots, r$ *we have* $(pk_{sig}, m^*) \neq (pk_{sig,j}, m'_j)$.

3.2 Immutability

The censor can use the Sanit algorithm to change message blocks which the signer declared as modifiable. If a malicious censor tries to modify other blocks this should not yield a correct signature. In the attack model below the malicious sanitizer $\mathcal{A}$ interacts with the signer to receive signatures σ_i for messages m_i, descriptions ADM_i and keys $pk_{\text{san},i}$ of its choice, before eventually outputting a valid pair $(pk^*_{\text{san}}, m^*, \sigma^*)$ such that message m^* is not a "legitimate" transformation of one of the m_i's under the same key $pk^*_{\text{san}} = pk_{\text{san},i}$. The latter is formalized by demanding that each m_i and m^* differ in at least one inadmissible block (or that $pk^*_{\text{san}} \neq pk_{\text{san},i}$).

Definition 3.2 (Immutability) *A sanitizable signature scheme* SanSig *is* immutable *if for any efficient algorithm $\mathcal{A}$ the probability that the following experiment returns 1 is negligible (as a function of n):*

Experiment $\mathsf{Immutability}^{\mathsf{SanSig}}_{\mathcal{A}}(n)$
$(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^n)$
$(pk^*_{san}, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{sig}, \cdot), \mathsf{Proof}(sk_{sig}, \dots, pk_{sig}, \cdot)}(pk_{sig})$
letting $(m_i, \text{ADM}_i, pk_{san,i})$ *and* σ_i *for* $i = 1, 2, \dots, q$
denote the queries and answers to and from oracle Sign.
return 1 *if*
$\mathsf{Verify}(m^*, \sigma^*, pk_{sig}, pk^*_{san}) = \texttt{true}$ *and*
for all $i = 1, 2, \dots, q$ *we have*
$pk^*_{san} \neq pk_{san,i}$, *or*
$m^*[j_i] \neq m_i[j_i]$ *for some* $j_i \notin \text{ADM}_i$
//where shorter messages are padded with blocks of the special symbol $\perp \notin \{0,1\}^*$

Thwarting Additional Sanitizing Attacks. Testifying to the fact that our definition is quite robust in the multi-party setting we discuss that our notion of immutability implies the "additional sanitizing attack" of Miyazaki et al. [MIM+05]. Suppose we have three parties in a department, the signer and two sanitizers. Both sanitizers are authorized in principle to modify messages, but for a specific message m only the first sanitizer is permitted to do so (say that this message contains information affecting the second sanitizer). Assume now that a requesting party asks for the non-sensitive parts of message m, and that the first sanitizer with public key pk_{san} is honest and changes the message m to derive a new signature σ' for m'. But now the second sanitizer with public key pk^*_{san} intercepts this reply, maliciously deletes the information about him in message m' and produces a signature σ^* for this bowdlerized message m^*. Only this pair m^*, σ^* is sent to the requesting party, looking like an authorized reply to the requesting party.

Our notion of immutability is strong enough to capture "additional sanitizing attacks" (assuming unique public keys of parties). Namely, in our definition we declare the adversary successful if it manages to find a new public key pk^*_{san} different from the sanitizer's public key pk_{san} such that the final output verifies correctly under this new key pk^*_{san}. An adversary can now mount the additional sanitizing attack by generating the keys of the honest sanitizer internally (in a sense, giving even more control to the adversary), calling the signer to create the document for the key pk_{san} of the honest sanitizer and then outputting the further censored message m^* with σ^* under a public key pk^*_{san}. Hence, immutability guarantees that such a case cannot succeed and, in particular, that the scheme is secure against "additional sanitizing attacks".

3.3 Privacy

Privacy roughly means that it should be infeasible to recover information about the sanitized parts of the message. As information leakage through the modified message itself can never be prevented, we only refer to information which is available through the sanitized signature. There are two possible flavors in formalizing privacy for sanitizable signatures. One approach follows semantic security of encryption schemes and is called *semantic privacy*. It says that for any adversary $\mathcal{A}$ seeing sanitized signatures there is a simulator $\mathcal{S}$ which is denied the signatures, but which is still as successful in predicting some information about the original message as $\mathcal{A}$. This notion is discussed comprehensively in Appendix A.

The other approach is based on the indistinguishability notion for encryption. In this case, an adversary can choose pairs (m_0, MOD_0), (m_1, MOD_1) of messages and modifications together with a description ADM and has access to a "left-or-right" box. This oracle either returns a sanitized signature for the left tuple $(b = 0)$ or for the right tuple $(b = 1)$. The task of the attacker is to predict the random bit b significantly better than by guessing. Here we need the additional constraint that for each call to the left-or-right box the resulting modified messages are identical for both tuples and the modifications both match ADM, else the task would be trivial. We write $(m_0, \text{MOD}_0, \text{ADM}) \equiv (m_1, \text{MOD}_1, \text{ADM})$ for this.

Below we formalize the more handy indistinguishability notion and discuss in Appendix A that the simulation-based approach is equivalent (as in case of encryption). In our definition of privacy we grant the adversary also access to a signature and a sanitizer oracle, enabling the adversary to create signatures which can be sanitized afterwards. We note that the adversary does not get to choose the signature $\sigma_{j,b}$ for inputs to the left-or-right box. Instead, this signature is first computed from scratch. This corresponds to the "hospital setting" mentioned in the introduction, where the medical data and, in particular, their signatures are kept confidentially and only the sanitized document is released. One may define a stronger version where the adversary gets to choose $\sigma_{j,0}, \sigma_{j,1}$, but it seems much harder to realize this requirement efficiently.

As in case of unforgeability and immutability we also grant the adversary access to Proof. Hence, since we let the adversary also determine the input to this box the adversary may input the data received from the Sign box here, but cannot use any of the initially computed and secret signatures in the calls to the left-or-right box (unless the adversary accidently guesses one). The reason is again that proofs usually leak information about the signatures but the signatures in the left-or-right box should remain secret (as in the hospital example).

Definition 3.3 (Privacy) *A sanitizable signature scheme* SanSig *is* private *if for any efficient algorithm* $\mathcal{A}$ *the probability that the following experiment returns* 1 *is negligibly close to* $\frac{1}{2}$*:*

Experiment $\mathsf{Privacy}^{\mathsf{SanSig}}_{\mathcal{A}}(n)$
$(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^n)$
$(pk_{san}, sk_{san}) \leftarrow \mathsf{KGen}_{san}(1^n)$
$b \leftarrow \{0,1\}$
$a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,sk_{sig},\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,sk_{san},\cdot),\mathsf{Proof}(sk_{sig},\cdots),\mathsf{LoRSanit}(\cdot,\cdot,\cdot,sk_{sig},sk_{san},b)}(pk_{sig}, pk_{san})$
where oracle $\mathsf{LoRSanit}(\cdot,\cdot,\cdot,sk_{sig},sk_{san},b)$
on input $(m_{j,0}, \mathrm{MOD}_{j,0}, (m_{j,1}, \mathrm{MOD}_{j,1})$ *and* ADM_j
first computes $\sigma_{j,b} \leftarrow \mathsf{Sign}(m_{j,b}, sk_{sig}, pk_{san}, \mathrm{ADM}_j)$ *and then*
returns $(m'_j, \sigma'_j) \leftarrow \mathsf{Sanit}(m_{j,b}, \mathrm{MOD}_{j,b}, \sigma_{j,b}, pk_{sig}, sk_{san})$,
and where $(m_{j,0}, \mathrm{MOD}_{j,0}, \mathrm{ADM}_j) \equiv (m_{j,1}, \mathrm{MOD}_{j,1}, \mathrm{ADM}_j)$,
i.e., are mapped to the same modified message.
return 1 *if* $a = b$.

3.4 Transparency

For transparency the original work of Ateniese et al. [ACdMT05] distinguishes between two notions, called weak and strong transparency. In the case of weak transparency an adversary, given a signed message m with a valid signature σ, should not be able to correctly guess whether m has been sanitized or was simply signed. In the case of strong transparency, the adversary should not even be able to tell which parts of the message are potentially mutable. Since the latter seems an overly strong requirement —observe that this implies that the information ADM must be hidden and must not be recoverable from σ, for example— we call weak transparency simply transparency here and formalize only this notion.

We define transparency by the following adversarial game. We consider an adversary $\mathcal{A}$ with access to Sign, Sanit and Proof oracles with which the adversary can create signatures for (sanitized) messages and learn proofs. In addition, $\mathcal{A}$ gets access to a Sanit/Sign box which contains a secret random bit $b \in \{0,1\}$ and which, on input a message m, a modification information MOD and a description ADM

- for $b = 0$ runs the signer to create $\sigma \leftarrow \mathsf{Sign}(m, sk_{\mathrm{sig}}, pk_{\mathrm{sig}}, \mathrm{ADM})$, then runs the sanitizer and returns the sanitized message m' with the new signature σ', and
- for $b = 1$ acts as in the case $b = 0$ but also signs m' from scratch with the signing algorithm to create a signature σ' and returns the pair (m', σ').

Adversary $\mathcal{A}$ eventually produces an output a, the guess for b. A sanitizable signature is now said to be *transparent* if for all efficient algorithms $\mathcal{A}$ the probability for a right guess $a = b$ in the above game is negligibly close to $\frac{1}{2}$.

Definition 3.4 (Transparency) *A sanitizable signature scheme* SanSig *is* transparent *if for any efficient algorithm* $\mathcal{A}$ *the probability that the following experiment returns* 1 *is negligibly close to* $\frac{1}{2}$*:*

Experiment $\mathsf{Transparency}^{\mathsf{SanSig}}_{\mathcal{A}}(n)$
$(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^n)$
$(pk_{san}, sk_{san}) \leftarrow \mathsf{KGen}_{san}(1^n)$
$b \leftarrow \{0,1\}$
$a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,sk_{sig},\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,sk_{san}),\mathsf{Proof}(sk_{sig},\dots),\mathsf{Sanit/Sign}(\cdot,\cdot,\cdot,sk_{sig},sk_{san},pk_{sig},pk_{san},b)}(pk_{sig}, pk_{san})$
where oracle $\mathsf{Sanit/Sign}$ *for input* $m_k, \mathrm{MOD}_k, \mathrm{ADM}_k$
computes $\sigma_k \leftarrow \mathsf{Sign}(m_k, sk_{sig}, pk_{san}, \mathrm{ADM}_k)$
then $(m'_k, \sigma'_k) \leftarrow \mathsf{Sanit}(m_k, \mathrm{MOD}_k, \sigma_k, pk_{sig}, sk_{san})$,
then, if $b = 1$*, replaces* σ'_k *by* $\sigma'_k \leftarrow \mathsf{Sign}(m'_k, sk_{sig}, pk_{san}, \mathrm{ADM}_k)$,
and finally returns (m'_k, σ'_k).
return 1 *if* $a = b$

We note that, analogously to the case of privacy, we have σ_k be created by the signer locally in the Sanit/Sign box. A stronger requirement would enable the adversary to determine this signature as part of the input. Yet, this notion again does not reflect the "hospital scenario" nor does it seem to be easy to realize efficiently. Similarly, the adversary cannot use these signatures in the Proof box.

Also note that, with the definition above, schemes with deterministic signature or sanitizing algorithms cannot be transparent, because an adversary could then easily compare answers from the Sanit/Sign box with outputs of the signature sanitizing oracle. Yet, since some applications may need transparency even if a message has been signed or sanitized before, we provide the stronger requirement. The weaker guarantee would then also demand from the adversary's queries to the signing and sanitizing boxes that for all k we have $m'_k \neq m_i$ for all i and $m'_k \neq m'_j$ for all j.

3.5 Accountability

Accountability says that the origin of a (sanitized) signature should be undeniable. There are two types of accountability:

Sanitizer-Accountability. If a message has not been signed by the signer, then even a malicious sanitizer should not be able to make the judge accuse the signer.

Signer Accountability. If a message and its signature have not been sanitized, then even a malicious signer should not be able to make the judge accuse the sanitizer.

Both notions are formalized below through two similar, yet slightly different adversarial games.

In the sanitizer-accountability game let $\mathcal{A}_{\mathsf{Sanit}}$ be an efficient adversary playing the role of the malicious sanitizer. Adversary $\mathcal{A}_{\mathsf{Sanit}}$ has access to a Sign oracle and a Proof oracle. Its task is to output a valid message-signature pair m^*, σ^* together with a key pk^*_{san} (with (pk^*_{san}, m^*) being different from messages previously signed by the Sign oracle) such that the proof produced by the signer via Proof still leads Judge to decided "Sig", i.e., that the signature has been created by the signer.

Definition 3.5 (Sanitizer-Accountability) *A sanitizable signature scheme* SanSig *is* sanitizer-accountable *if for any efficient algorithm $\mathcal{A}_{Sanit}$ the probability that the following experiment returns* 1 *is negligible (as a function of n):*

Experiment $\mathit{San\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{Sanit}}(n)$
$(pk_{sig}, sk_{sig}) \leftarrow \mathit{KGen}_{sig}(1^n)$
$(pk^*_{san}, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathit{Sign}(\cdot, sk_{sig}, \cdot, \cdot), \mathit{Proof}(sk_{sig}, \dots)}_{\mathsf{Sanit}}(pk_{sig})$
letting $(m_i, \text{ADM}_i, pk_{san,i})$ *and* σ_i *for* $i = 1, 2, \dots, q$
denote the queries and answers to and from oracle Sign
$\pi \leftarrow \mathit{Proof}(sk_{sig}, m^*, \sigma^*, (m_1, \sigma_1), \dots, (m_q, \sigma_q), pk_{san})$
return 1 *if*
$(pk^*_{san}, m^*) \neq (pk_{san,i}, m_i)$ *for all* $i = 1, 2, \dots, q$, *and*
$\mathit{Verify}(m^*, \sigma^*, pk_{sig}, pk^*_{san}) = \mathtt{true}$, *and*
$\mathit{Judge}(m^*, \sigma^*, pk_{sig}, pk^*_{san}, \pi) = \mathtt{Sig}$

In the signer-accountability game a malicious signer $\mathcal{A}_{\mathsf{Sign}}$ gets a public sanitizing key pk_{san} as input. It is allowed to query a sanitizing oracle about tuples $(m_i, \text{MOD}_i, \sigma_i, pk_{\text{sig},i})$ receiving answers (m'_i, σ'_i). Adversary $\mathcal{A}_{\mathsf{Sign}}$ finally outputs a tuple $(pk^*_{\text{sig}}, \pi^*, m^*, \sigma^*)$ and is considered to succeed if Judge accuses the sanitizer for the new key-message pair pk^*_{sig}, m^* with a valid signature σ^*. Note that our model allows the proof π to contain information about the original message.

Definition 3.6 (Signer-Accountability) *A sanitizable signature scheme* SanSig *is* signer-accountable *if for any efficient algorithm $\mathcal{A}_{Sign}$ the probability that the following experiment returns* 1 *is negligible (as a function of n):*

Experiment $\mathit{Sig\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{\mathit{Sign}}}(n)$
$(pk_{san}, sk_{san}) \leftarrow \mathit{KGen}_{san}(1^n)$
$(pk^*_{sig}, \pi^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathit{Sanit}(\cdot,\cdot,\cdot,\cdot,sk_{san})}_{\mathit{Sign}}(pk_{san})$
letting (m'_i, σ'_i) *for* $i = 1, 2, \ldots, q$
denote the answers from oracle Sanit.
return 1 *if*
$(pk^*_{sig}, m^*) \neq (pk_{sig,i}, m'_i)$ *for all* $i = 1, 2, \ldots, q$, *and*
$\mathit{Verify}(m^*, \sigma^*, pk^*_{sig}, pk_{san}) = \mathtt{true}$ *and*
$\mathit{Judge}(m^*, \sigma^*, pk^*_{sig}, pk_{san}, \pi^*) = \mathtt{San}$

4 Relationships of the Security Requirements

In this section we show that except for the privacy and the unforgeability requirement all other notions are independent (in the sense that none of them follows from the other properties, even if they all hold at the same time). We first show that privacy follows from transparency alone, and unforgeability holds if the two versions of accountability hold simultaneously. We then show the independence of the other requirements.

We stress again that our results are in contrast to the claim by Ateniese et al. [ACdMT05] that, for example, accountability follows from the unforgeability requirement. Our results show that unforgeability follows from accountability whereas the other direction is not true. It is not clear if Ateniese et al. [ACdMT05] consider signer-accountability at all, or merely refer to sanitizer-accountability. However, as we have argued both versions of accountability are desirable to avoid framing attacks from either side, and in either case we also show that sanitizer-accountability alone does not imply unforgeability.

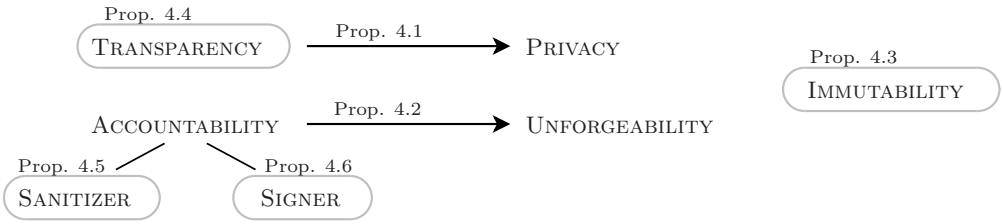


Figure 2: Summary of the relations among the security properties of sanitizable signatures. Arrows represent implications, frames represent the independence from other requirements.

4.1 Implications

We show that privacy follows from transparency. The idea is that for a transparent scheme one cannot distinguish between signatures created by the signer and ones produced by the sanitizer. Hence, we can essentially replace the left-or-right sanitizing oracle in the privacy experiment by the procedure which creates the signatures for the sanitized message with the help of the signer algorithm. But since the privacy experiment requires the sanitized messages to be identical, the answer is always a fresh signature for the same message, independent of the left-or-right question, and privacy follows.

Proposition 4.1 (Transparency Implies Privacy) *A transparent sanitizable signature scheme is also private.*

Proof. We show this through contraposition, i.e., we construct a successful attacker against transparency assuming a successful attacker against privacy. Let $\mathcal{A}_{\text{priv}}$ be an attacker against privacy. We construct $\mathcal{A}_{\text{trans}}$, essentially running a black-box simulation of $\mathcal{A}_{\text{priv}}$, as follows.

$\mathcal{A}_{\text{trans}}$ gets pk_{sig} and pk_{san} as input, which it forwards to $\mathcal{A}_{\text{priv}}$. Furthermore, $\mathcal{A}_{\text{trans}}$ picks a random bit $b^* \leftarrow \{0,1\}$. For every value that $\mathcal{A}_{\text{priv}}$ requests to be signed, sanitized or proven $\mathcal{A}_{\text{trans}}$ forwards the corresponding value to its own external box and hands the answer back to $\mathcal{A}_{\text{priv}}$. For every pair $(m_0, \text{MOD}_0), (m_1, \text{MOD}_1)$ with ADM that $\mathcal{A}_{\text{priv}}$ sends to the left-or-right sanitizer box, $\mathcal{A}_{\text{trans}}$ forwards message $(m_{b^*}, \text{MOD}_{b^*}, \text{ADM})$ to its $\mathsf{Sanit/Sign}$ box. The answer from the the box is returned to $\mathcal{A}_{\text{priv}}$. Eventually, $\mathcal{A}_{\text{priv}}$ outputs its guess a. Algorithm $\mathcal{A}_{\text{trans}}$ outputs $a^* = 0$ iff $a = b^*$.

We now look at the probabilities of $\mathcal{A}_{\text{trans}}$ being successful:

- Given that $b = 0$ (i.e., the $\mathsf{Sanit/Sign}$ box always sanitizes) the simulation from $\mathcal{A}_{\text{priv}}$'s point of view is identical to the actual attack against privacy (with a random bit b^*). Hence,

$$\text{Prob}[\, a^* = 0 \mid b = 0\,] = \text{Prob}[\, \mathcal{A}_{\text{priv}} = b^*\,].$$

- Given that $b = 1$ (i.e., the $\mathsf{Sanit/Sign}$ box always signs the modified message) the bit b^* is information-theoretically hidden from $\mathcal{A}_{\text{priv}}$ (observing that the modified messages from queries in the privacy experiment have to be identical and the input to the signing algorithm is therefore independent of b^*):

$$\text{Prob}[\, a^* = 1 \mid b = 1\,] = \tfrac{1}{2}.$$

Since we have assumed $\mathcal{A}_{\text{priv}}$ to be successful, $\mathcal{A}_{\text{priv}}$ has at least a success probability of $\frac{1}{2} + \frac{1}{\text{poly}(n)}$ for some polynomial poly(n). It follows that $\mathcal{A}_{\text{trans}}$ has a success probability of at least

$$\begin{aligned}
\text{Prob}[\, \mathcal{A}_{\text{trans}} = b] &= \text{Prob}[b = 0] \cdot \text{Prob}[\, \mathcal{A}_{\text{trans}} = 0 \mid b = 0] \\
&\quad + \text{Prob}[b = 1] \cdot \text{Prob}[\, \mathcal{A}_{\text{trans}} = 1 \mid b = 1] \\
&= \frac{1}{2} \cdot \left(\frac{1}{2} + \frac{1}{\text{poly}(n)} \right) + \frac{1}{2} \cdot \frac{1}{2} \\
&= \frac{1}{2} + \frac{1}{2 \cdot \text{poly}(n)}
\end{aligned}$$

which is significantly larger than $\frac{1}{2}$. □

As mentioned above, unforgeability is implied by the two versions of accountability. The idea behind the result is that, given a successful forgery, the judge cannot really decide if this forgery has been produced by the signer or the sanitizer. Else the judge was biased towards outputting `Sig` or `San` for indecisive cases too often, contradicting either the sanitizer- or signer-accountability.

Proposition 4.2 (Accountability Implies Unforgeability) *A sanitizable signature scheme which is sanitizer-accountable and signer-accountable is also unforgeable.*

Proof. We show this proposition again by contraposition, constructing attackers $\mathcal{A}_{\mathsf{Sanit}}$ and $\mathcal{A}_{\mathsf{Sign}}$ against the versions of accountability assuming a successful attacker $\mathcal{A}_{\text{unf}}$ against unforgeability. We first describe the adversary $\mathcal{A}_{\mathsf{Sanit}}$ against sanitizer-accountability, running the corresponding accountability experiment. The idea behind our adversaries is to use adversary $\mathcal{A}_{\text{unf}}$ as a subroutine, creating a virtual environment matching the unforgeability experiment. The resulting attackers will both be efficient.

More formally, adversary $\mathcal{A}_{\mathsf{Sanit}}$ receives as input a public key pk_{sig} of the signer. It runs the key generation algorithm $\mathsf{KGen}_{\text{san}}(1^n)$ to derive a key pair $(pk_{\text{san}}, sk_{\text{san}})$ and initializes a black-box simulation of adversary $\mathcal{A}_{\text{unf}}$ on $(pk_{\text{sig}}, pk_{\text{san}})$. For each subsequent signature or proof request of the unforgeability adversary, our attacker $\mathcal{A}_{\mathsf{Sanit}}$ forwards the requests to its own oracle and hands the reply back to $\mathcal{A}_{\text{unf}}$. For each sanitizing request our adversary exploits the knowledge of sk_{san} and proceeds as the sanitizing algorithm would. When $\mathcal{A}_{\text{unf}}$ finally outputs the forgery attempt m^*, σ^* then $\mathcal{A}_{\mathsf{Sanit}}$, too, outputs $(pk_{\text{san}}, m^*, \sigma^*)$ and stops.

Before analyzing adversary $\mathcal{A}_{\mathsf{Sanit}}$ we first describe our attacker $\mathcal{A}_{\mathsf{Sign}}$ against signer-accountability. This adversary operates essentially in the same way as $\mathcal{A}_{\mathsf{Sanit}}$, mounting a black-box simulation of $\mathcal{A}_{\text{unf}}$. The

only exception lies in the role switch of the signature and sanitizing steps, i.e., $\mathcal{A}_{\mathsf{Sign}}$ simulates Sign and Proof internally (with self-chosen keys $sk_{\text{sig}}, pk_{\text{sig}}$) and uses its external Sanit oracle to answer such queries. Adversary $\mathcal{A}_{\mathsf{Sign}}$ also makes an additional step at the end. Having received m^*, σ^* from $\mathcal{A}_{\text{unf}}$ it runs algorithm Proof on $sk_{\text{sig}}, m^*, \sigma^*$, all previously signed message-signature pairs and the public keys to derive π^*. Our attacker then outputs $(pk_{\text{sig}}, \pi^*, m^*, \sigma^*)$.

For the analysis let $\text{FORGE}_{\mathsf{Sanit}}$ and $\text{FORGE}_{\mathsf{Sign}}$ be the events that $\mathcal{A}_{\text{unf}}$ in the simulation of $\mathcal{A}_{\mathsf{Sanit}}$ resp. $\mathcal{A}_{\mathsf{Sign}}$ above outputs m^*, σ^* such that m^* has never been submitted to the signing oracle nor been the answer of the sanitizing oracle (or, if so, a different public key has been used in the query), and that the conclusive run of the verifier in the corresponding accountability experiment returns `true`. Note that since the simulations are identical we can drop the subscript and simply speak of event FORGE. Also note that the simulations are perfect in the sense that they match the unforgeability experiment ideally, and the probability for event FORGE to happen is exactly the probability of $\mathcal{A}_{\text{unf}}$ winning in the unforgeability experiment.

We clearly have

$$\begin{aligned}
&\text{Prob}\Big[\,\mathsf{San\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{\mathsf{Sanit}}}(n) = 1\Big] \\
&\qquad \geq \quad \text{Prob}\Big[\,\mathsf{San\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{\mathsf{Sanit}}}(n) = 1 \;\Big|\; \text{FORGE}\Big] \cdot \text{Prob}[\,\text{FORGE}] \\
&\text{Prob}\Big[\,\mathsf{Sig\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{\mathsf{Sign}}}(n) = 1\Big] \\
&\qquad \geq \quad \text{Prob}\Big[\,\mathsf{Sig\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{\mathsf{Sign}}}(n) = 1 \;\Big|\; \text{FORGE}\Big] \cdot \text{Prob}[\,\text{FORGE}]
\end{aligned}$$

Conditioning on FORGE we note that the probability for $\mathsf{Sig\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{\mathsf{Sign}}}(n) = 1$, i.e., that Judge returns Sanit, is exactly the probability for $\mathsf{San\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{\mathsf{Sanit}}}(n) = 0$, i.e., Judge not outputting Sign. The experiments only differ in two points:

- In $\mathcal{A}_{\mathsf{Sanit}}$'s attack the Proof algorithm is called externally by the experiment, in $\mathcal{A}_{\mathsf{Sign}}$'s case it is called internally. But both times Proof is executed on genuine data.
- In the sanitizer-accountability experiment we require (pk^*_{san}, m^*) to be distinct from all queries to the signature oracle, whereas in the signer-accountability experiment (pk^*_{sig}, m^*) must be different from all answers from the sanitizing oracle. But since we assume FORGE both conditions are satisfied simultaneously.

Taking the above into account and adding the two terms we thus obtain:

$$\begin{aligned}
&\text{Prob}\Big[\,\mathsf{San\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{\mathsf{Sanit}}}(n) = 1\Big] + \text{Prob}\Big[\,\mathsf{Sig\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{\mathsf{Sign}}}(n) = 1\Big] \\
&\quad \geq \quad \text{Prob}\Big[\,\mathsf{San\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{\mathsf{Sanit}}}(n) = 1 \;\Big|\; \text{FORGE}\Big] \cdot \text{Prob}[\,\text{FORGE}] \\
&\qquad\qquad + \big(1 - \text{Prob}\Big[\,\mathsf{San\text{-}Accountability}^{\mathsf{SanSig}}_{\mathcal{A}_{\mathsf{Sanit}}}(n) = 1 \;\Big|\; \text{FORGE}\Big]\,\big) \cdot \text{Prob}[\,\text{FORGE}] \\
&\quad = \quad \text{Prob}[\,\text{FORGE}]\,.
\end{aligned}$$

By assumption the latter probability is non-negligible, implying that at least one of the probabilities and the left hand side must be non-negligible, too. This, however, contradicts (at least) one of the accountabilities notions. □

In Appendix C we show that one of the accountability properties alone does not suffice to imply unforgeability, even when assuming immutability and transparency (and thus privacy) as well.

4.2 Separations

Now we show that all the other security requirements are independent, i.e., no property follows from a combination of the other properties. Our results all assume that there exist secure sanitizable signature

scheme obeying all properties (which, according to the next section, exist under common cryptographic assumptions) and then show that there is a scheme inheriting all properties except for the one in question. We start with immutability:

Proposition 4.3 (Independence of Immutability) *Assume that there exists a secure sanitizable signature scheme. Then there exists a sanitizable signature scheme which is unforgeable, private, transparent, sanitizer-accountable and signer-accountable, but not immutable.*

Proof. We show this proposition by modifying a sanitizable signature scheme fulfilling all security requirements in a way that immutability no longer holds, but all the other requirements remain unaffected. Let $\mathsf{SanSig} = (\mathsf{KGen}_{\mathrm{sig}}, \mathsf{KGen}_{\mathrm{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ be a secure sanitizable signature scheme and let $\mathsf{SanSig}' = (\mathsf{KGen}_{\mathrm{sig}}, \mathsf{KGen}'_{\mathrm{san}}, \mathsf{Sign}', \mathsf{Sanit}', \mathsf{Verify}', \mathsf{Proof}', \mathsf{Judge}')$ be the following modification of SanSig:

- $\mathsf{KGen}_{\mathrm{sig}}$ remains unchanged.
- $\mathsf{KGen}'_{\mathrm{san}}$ works as $\mathsf{KGen}_{\mathrm{san}}$, except that it concatenates a '1' to the generated public key, $pk'_{\mathrm{san}} = pk_{\mathrm{san}}||1$.
- Verify' outputs `true` if pk'_{san} ends with '0', else it cuts off the last bit of pk'_{san} and runs Verify for the input data and this shortened key.
- Judge' takes the public key of the sanitizer and outputs `San` if this public key ends with '0'. Else, it deletes this bit and runs the original Judge algorithm for the input data and this shortened key.
- all other algorithms merely chop off the last bit of pk'_{san} and work as their ancestors (but using the shortened key).

Since the adversary against immutability can choose a public sanitizing key ending with '0', in this modified scheme he can easily produce a positive verification for any message (without even requiring an original signature). Therefore, he is able to change forbidden blocks within a chosen message keeping the verification status `true`. Hence SanSig' is obviously not immutable.

Now consider the other properties. Since the key generation algorithm never outputs an exceptional key pk_{san} ending with '0', completeness is preserved. Also note that, except for the attackers against immutability and sanitizer-accountability, none of the adversaries in their games is able to choose the public sanitizing key pk'_{san}. Therefore, as the other algorithms basically only remove the final bit of the key and then proceed as before, the other security requirements clearly stay intact.

On the other hand, an attacker against sanitizer-accountability cannot benefit from choosing a key pk_{san} ending with '0' as the judge always accuses the sanitizer for such keys. For all other keys the sanitizer-accountability easily transfers from the original scheme. In summary, this scheme SanSig' fulfills all the security requirements except for immutability. □

Next we show that transparency is also independent:

Proposition 4.4 (Independence of Transparency) *Assume that there exists a secure sanitizable signature scheme. Then there exists a sanitizable signature scheme which is immutable, unforgeable, private, sanitizer-accountable and signer-accountable, but not transparent.*

Proof. Again let $\mathsf{SanSig} = (\mathsf{KGen}_{\mathrm{sig}}, \mathsf{KGen}_{\mathrm{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ be a sanitizable signature scheme fulfilling all security requirements and $\mathsf{SanSig}' = (\mathsf{KGen}_{\mathrm{sig}}, \mathsf{KGen}_{\mathrm{san}}, \mathsf{Sign}', \mathsf{Sanit}', \mathsf{Verify}', \mathsf{Proof}', \mathsf{Judge}')$ the following modification:

- algorithms $\mathsf{KGen}_{\mathrm{sig}}$ and $\mathsf{KGen}_{\mathrm{san}}$ stay unchanged.
- Sign' works as Sign, except that it appends '1' to the signature, $\sigma' = \sigma||1$.
- Sanit' works as Sanit, except that appends '0' to the signature, $\sigma' = \sigma||0$.

- algorithms Verify', Judge' and Proof' all delete the final bit of the input signature σ' (and in case of Proof' of all additional signature given as input) and then run the original algorithm on the input data and this shortened signature(s).

Since an adversary playing against transparency can simply output the last bit of σ' as its guess for the secret bit b, and wins with probability 1, the modified scheme is obviously not transparent. It remains to show that SanSig' fulfills the other requirements.

First note that completeness is not affected by the signature extension and shortening and follows from the completeness of the underlying scheme. Next, since the Verify, Judge and Proof algorithms basically ignore the final bit of signatures, immutability, unforgeability,and both versions of accountability remain secure. Formally, one can easily turn a successful adversary against any of the properties of the modified scheme into one against the corresponding property of the original scheme by appending and removing the bits in the signatures for oracle queries (except for the transparency experiments it is always publicly known whether one expects the signature to end with '0' or with '1'). This is also true for privacy where each signature of the sanitizing box merely contains the extra '0' bit. □

Finally we show that independence also holds for the two types of accountability:

Proposition 4.5 (Independence of Sanitizer-Accountability) *Assume that there exists a secure sanitizable signature scheme. Then there exists a sanitizable signature scheme which is immutable, unforgeable, private, transparent and signer-accountable, but not sanitizer-accountable.*

Proof. This time, we need an additional cryptographic primitive, namely a one-way function f. Note that the existence of an unforgeable sanitizable scheme already implies the existence of one-way functions, such that our solution does not rely on any extra assumption. We also assume that the block length t equals n and that message blocks are from $\{0,1\}^t$ (relaxing these requirements is possible but this simplifies the presentation here). Again, let $\mathsf{SanSig}' = (\mathsf{KGen}'_{\text{sig}}, \mathsf{KGen}'_{\text{san}}, \mathsf{Sign}', \mathsf{Sanit}', \mathsf{Verify}', \mathsf{Proof}', \mathsf{Judge}')$ be the following modification of a scheme SanSig having all properties:

- $\mathsf{KGen}'_{\text{sig}}$ runs $\mathsf{KGen}_{\text{sig}}$ to generate a key pair $(sk_{\text{sig}}, pk_{\text{sig}})$, It also picks a random $x \leftarrow \{0,1\}^n$ and outputs the modified secret key $sk'_{\text{sig}} = (sk_{\text{sig}}, x)$ and the modified public key $(pk_{\text{sig}}, f(x))$.
- $\mathsf{KGen}'_{\text{san}}$ works as $\mathsf{KGen}_{\text{san}}$, except that it concatenates '1' to the public key, $pk'_{\text{san}} = pk_{\text{san}}||1$.
- Sign' for input m, ADM and keys $sk'_{\text{sig}}, pk'_{\text{san}}$ first runs Sign for this input (with sk_{sig} and the last bit of pk'_{san} chopped off) to get a signature σ. It then checks if m is a single and modifiable block and that pk'_{san} consists of n zeros, in particular, that it ends with '0'. If so, it appends the preimage of the value $f(x)$ in the public key and returns (σ, x), else it outputs σ.
- Verify' outputs `true` if pk'_{san} consists of n zeros (and ends with '0') and its input message m consists of a single mutable block[1], and if m is a preimage of the value $f(x)$ in the public key pk'_{sig}. In all other cases it cuts off the last bit of pk'_{san} and executes Verify on the input data (and the shortened public key).
- Judge' outputs Sign if pk'_{san} ends with '0', else cuts off the last bit of pk'_{san} and runs Judge.
- all other algorithms cut off the last bit of pk'_{san} and execute their corresponding ancestor (but using the shortened key).

The adversary against sanitizer-accountability can easily produce a key pk^*_{san} consisting only of n bits '0' in the first stage. Then it picks a random single-block message m and submits this message together with the description ADM that this message is modifiable and pk^*_{san} to the signature oracle to receive a signature σ and the preimage x. The adversary outputs pk^*_{san}, $m^* = x$ and σ. It is clear that the modified verifier accepts

[1] As ADM is recoverable from the signature, Verify' can access this information.

this pair and that, except with probability 2^{-n} with which $m = m^*$, this is a new message. Moreover, Judge' outputs `Sig` for this key pk^*_{san}, and SanSig' thus does not fulfill sanitizer-accountability.

As for the other properties, completeness, unforgeability, transparency, privacy and signer-accountability cannot run into the exception mode, because honestly generated keys pk'_{san} do not end with '0'. It remains to show that the requirement of immutability remains true. There are two cases:

- If an adversary against immutability at some point receives x from the signature oracle as additional output then it has queried the oracle about a single-block, modifiable message m and has a public sanitizer key consisting of '0'. In this case, outputting some $pk^*_{\text{san}}, m^*, \sigma^*$ triggering the exceptional mode of the verifier, returning `true` for $m^* = x$, is not a successful attack against immutability anymore (because pk^*_{san} needs to be the all-zero string and m^* cannot differ from m in a forbidden block). In any other case the immutability from the original scheme carries over.
- If an adversary against immutability never receives x from the signer as additional output, it can only abuse the exceptional mode of the verifier if it is able to invert the one-way function f on the unknown input x (this informal argument can be easily turned into a formal inversion algorithm). Since this happens only with negligible probability, the immutability property follows from the immutability of the underlying scheme.

This shows that immutability is preserved. □

Proposition 4.6 (Independence of Signer-Accountability) *Assume that there exists a secure sanitizable signature scheme. Then there exists a sanitizable signature scheme which is immutable, unforgeable, private, transparent and sanitizer-accountable, but not signer-accountable.*

Proof. In the same way as above let $\mathsf{SanSig} = (\mathsf{KGen}_{\text{sig}}, \mathsf{KGen}_{\text{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ be a sanitizable signature scheme fulfilling all security requirements. Define the modified scheme $\mathsf{SanSig}' = (\mathsf{KGen}'_{\text{sig}}, \mathsf{KGen}_{\text{san}}, \mathsf{Sign}', \mathsf{Sanit}', \mathsf{Verify}', \mathsf{Proof}', \mathsf{Judge}')$ as follows:

- $\mathsf{KGen}_{\text{san}}$ remains unchanged.
- $\mathsf{KGen}'_{\text{sig}}$ works as $\mathsf{KGen}_{\text{sig}}$, except that it concatenates '1' to the signer key, $pk'_{\text{sig}} = pk_{\text{sig}}||1$.
- Verify' outputs `true` if pk'_{sig} ends with '0', else it cuts off the last bit of pk'_{sig} and runs Verify on the input data and the shortened key.
- Judge' outputs `San` if pk'_{sig} ends with '0', else it cuts off the last bit of pk'_{sig} and runs Judge on the input data and the shortened key.
- all other algorithms cut off the last bit of pk'_{sig} and work as their ancestors (but using the shortened key as input).

Since the adversary against signer-accountability can easily choose a public signing key ending with '0', he can easily produce an output that verifies and is still judged `San`. Thus, SanSig' does not fulfill signer-accountability. Since in all other cases the public signer key is chosen genuinely, the exception cases do not apply and all other properties (including completeness) are inherited. □

5 Sanitizable Signatures based on Chameleon Hashes

In this section we show that our security requirements can be met. Our construction is a modification of the scheme by Ateniese et al. [ACdMT05] and also uses chameleon hashes. The idea is as follows: Instead of signing the full message in clear we first replace modifiable message blocks $m[i]$ by (randomized) hash values $h[i] = \mathsf{CHash}(pk_{\text{san}}, m[i]; r[i])$ of the blocks. Then we sign this sequence of message blocks and hash values with a regular signature scheme.

The hash values have the special "chameleon" property that, if one has the sanitizer's trapdoor information sk_{san} and $r[i]$, one can easily find collisions, i.e., for given $m'[i]$ one is able to determine $r'[i]$ with $h[i] = \mathsf{CHash}(pk_{\text{san}}, m'[i]; r'[i])$, leaving the hash value invariant. This allows the sanitizer to modify message blocks for which the signer includes the $r[i]$'s in the signature (and only those), and such that the actual signature on the hash values does not need to be modified. We note that implementing the idea is more complicated due to the accountability problem, requiring something related to (but not exactly like) key-exposure freeness [AdM04] from the chameleon hash. The latter also necessitates the usage of tags entering the hash computations.

5.1 Construction

A chameleon hash scheme $\mathcal{CH} = (\mathsf{CHKGen}, \mathsf{CHash}, \mathsf{CHAdapt})$ (with tags) consists of three efficient algorithms such that algorithm CHKGen on input 1^n returns a key pair (sk, pk), algorithm CHash on input pk, a tag $\text{TAG} \in \{0,1\}^n$, a message m and randomness r (which is efficiently samplable from some range $\mathcal{R}_{pk}$) returns a hash value $h = \mathsf{CHash}(pk, \text{TAG}, m; r)$ and algorithm $\mathsf{CHAdapt}$ on input sk, TAG, m, r and TAG', m' returns r' such that $\mathsf{CHash}(pk, \text{TAG}, m; r) = \mathsf{CHash}(pk, \text{TAG}', m'; r')$. It also holds that for any $pk, \text{TAG}, m, \text{TAG}', m'$ the distribution of $\mathsf{CHAdapt}(sk, \text{TAG}, m, r, \text{TAG}', m')$ (over the choice of r) is the same as the distribution of r itself, also implying that a hash value $\mathsf{CHash}(pk, \text{TAG}, m; r)$ (over the choice of r) is distributed independently of TAG, m.

Key-exposure freeness [AdM04] now says that it is infeasible to find collisions, even if one gets to see collisions for other values. To be more precise, the security requirement demands that, after having learned collisions for some tags, one cannot create a collision for a new tag. This is a strong and useful notion and, yet, it would not be sufficient to provide security in our setting. Suppose we attach tags to the documents such that the signer modifies messages by finding collisions for the hash value for the corresponding tags. Then a malicious signer could still try to escape accountability by finding further collisions *for the same tag*. We therefore introduce the notion of collision-resistance *under random-tagging attacks*, i.e., where collisions for different tags are created but where one of the two tags is chosen at random (and the other one is provided by the adversary). In Appendix D we show that such chameleon hashes exist under the RSA assumption in the random oracle model:

Definition 5.1 (Collision-Resistance under Random-Tagging Attacks) *A chameleon hash scheme $\mathcal{CH} = (\mathsf{CHKGen}, \mathsf{CHash}, \mathsf{CHAdapt})$ is* collision-resistant under random-tagging attacks *if for any efficient adversary $\mathcal{A}$ the following experiment returns 1 with negligible probability only:*

Experiment $\mathsf{RndTag}^{\mathcal{CH}}_{\mathcal{A}}(n)$
$(pk, sk) \leftarrow \mathsf{CHKGen}(1^n)$
$(\text{TAG}, m, r, \text{TAG}', m', r') \leftarrow \mathcal{A}^{\mathsf{OAdapt}(sk,\cdot,\cdot,\cdot,\cdot)}(pk)$
where oracle OAdapt *for the i-th query* $(\text{TAG}_i, m_i, r_i, m'_i)$ *with* $\text{TAG}_i \in \{0,1\}^n$
samples $\text{TAG}'_i \leftarrow \{0,1\}^n$ *and computes* $r'_i \leftarrow \mathsf{CHAdapt}(sk, \text{TAG}, m, r, \text{TAG}', m')$.
Return (TAG'_i, r'_i).
return 1 *if*
$(\text{TAG}, m) \neq (\text{TAG}', m')$ *and*
$\mathsf{CHash}(pk, \text{TAG}, m; r) = \mathsf{CHash}(pk, \text{TAG}', m'; r')$ *and*
$\{(\text{TAG}, m), (\text{TAG}', m')\} \neq \{(\text{TAG}_i, m_i), (\text{TAG}'_i, m'_i)\}$ *for all* $i = 1, 2, \ldots$ *and*
$\{(\text{TAG}, m), (\text{TAG}', m')\} \neq \{(\text{TAG}'_i, m'_i), (\text{TAG}'_j, m'_j)\}$ *for all* $i, j = 1, 2, \ldots$.

The condition $\{(\text{TAG}, m), (\text{TAG}', m')\} \neq \{(\text{TAG}_i, m_i), (\text{TAG}'_i, m'_i)\}$ rules out trivial duplication attacks in which the adversary simply copies the data from the interaction with the oracle. The other condition $\{(\text{TAG}, m), (\text{TAG}', m')\} \neq \{(\text{TAG}'_i, m'_i), (\text{TAG}'_j, m'_j)\}$ prevents trivial "transitivity" attacks where the adversary calls the oracle about the same (TAG_i, m_i, r_i) twice, but with different m'_i, m'_j. Then the oracle's answers collide, as they yield the same value $\mathsf{CHash}(pk, \text{TAG}_i, m_i; r_i)$ individually.

In our construction we also need that the tags generated by the signer and the ones by the sanitizer look identical (from the outside) but are generated differently (and that this is provable to a judge). Otherwise

a malicious signer would be able to claim that a sanitized message has been the original. We resolve this by letting the tags of the sanitizer be truly random, whereas the tags of the signer need to be created pseudorandomly (with a pseudorandom generator PRG mapping n-bit inputs to $2n$-bit outputs). In addition, the seed for the pseudorandomly generated labels should be recoverable for the signer from the signature and the secret key, such that we use a pseudorandom function PRF (mapping n-bit inputs to n-bit outputs for n-bit keys) to derive the seed for PRG from a nonce NONCE, included in the signature.

Finally, we also need a regular signature scheme $\mathcal{S} = (\mathsf{SKGen}, \mathsf{SSign}, \mathsf{SVf})$ being existentially unforgeable under adaptive chosen-message attacks. Below we let $(a_1, a_2, \dots)$ be some encoding of bit strings $a_1, a_2, \dots$ into $\{0,1\}^*$ such that (in contrast to concatenation $a_1||a_2||\dots$) all individual components are recoverable:

Construction 5.2 (Sanitizable Signature Scheme) *Define the following sanitizable signature scheme* $\mathsf{SanSig} = (\mathsf{KGen}_{sig}, \mathsf{KGen}_{san}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$*:*

KEY GENERATION. *Algorithm* KGen_{sig} *on input* 1^n *generates a key pair* $(pk, sk) \leftarrow \mathsf{SKGen}(1^n)$ *of the underlying signature scheme, picks a key* $\kappa \leftarrow \{0,1\}^n$ *for the pseudorandom function and returns* $(pk_{sig}, sk_{sig}) = (pk, (sk, \kappa))$*. Algorithm* KGen_{san} *for input* 1^n *returns a pair* $(pk_{san}, sk_{san}) \leftarrow \mathsf{CHKGen}(1^n)$ *of the chameleon hash scheme.*

SIGNING. *Algorithm* Sign *on input* $m \in \{0,1\}^{t\ell}$, sk_{sig}, pk_{san}, ADM *picks* NONCE $\leftarrow \{0,1\}^n$ *at random, computes* $x = \mathsf{PRF}(\kappa, \text{NONCE})$ *and* TAG $= \mathsf{PRG}(x)$*, and picks* $r[j]$ *for each* j *in* ADM *at random. It computes*

$$h[j] = \begin{cases} \mathsf{CHash}(pk_{san}, \text{TAG}, (j, m[j], pk_{sig}); r[j]) & \text{if } j \text{ is in ADM} \\ m[j] & \text{else} \end{cases}$$

for each block $m[j] \in \{0,1\}^t$ *and computes* $\sigma_0 \leftarrow \mathsf{SSign}(sk_{sig}, (h, pk_{san}, \text{ADM}))$ *for* $h = (h[1], h[2], \dots, h[\ell])$*. It returns* $\sigma = (\sigma_0, \text{TAG}, \text{NONCE}, \text{ADM}, r[j_1], \dots, r[j_k])$ *where each* j_i *is in* ADM*.*

SANITIZING. *Algorithm* Sanit *on input a message* m*, information* MOD*, a signature* $\sigma = (\sigma_0, \text{TAG}, \text{NONCE}, \text{ADM}, r[j_1], \dots, r[j_k])$*,* pk_{sig} *and* sk_{san} *first checks that each modification in* MOD *is admissible according to* ADM *and that* σ_0 *is a valid signature for* $(h, pk_{san}, \text{ADM})$*. If not, it stops with output* $\perp$*. Else, for each* j *in* ADM *it lets* $m'[j]$ *be the modified block of* $m[j]$ *(possibly* $m'[j] = m[j]$*), picks new values* $\text{NONCE}' \leftarrow \{0,1\}^n$ *and* $\text{TAG}' \leftarrow \{0,1\}^{2n}$ *and replaces each* $r[j]$ *in the signature by*

$$r'[j] \leftarrow \mathsf{CHAdapt}(sk_{san}, \text{TAG}, (j, m[j], pk_{sig}), r[j], \text{TAG}', (j, m'[j], pk_{sig})).$$

It outputs m' *and* $\sigma' = (\sigma_0, \text{TAG}', \text{NONCE}', \text{ADM}, r'[j_1], \dots, r'[j_k])$*.*

VERIFICATION. *Algorithm* Verify *on input a message* $m \in \{0,1\}^{t\ell}$ *and a signature* $\sigma = (\sigma_0, \text{TAG}, \text{NONCE}, \text{ADM}, r[i_1], \dots, r[i_k])$*,* pk_{sig} *and* pk_{san} *computes*

$$h[j] = \begin{cases} \mathsf{CHash}(pk_{san}, \text{TAG}, (j, m[j], pk_{sig}); r[j]) & \text{if } j \text{ is in ADM} \\ m[j] & \text{else} \end{cases}$$

and then outputs $\mathsf{SVf}(pk_{san}, (h, pk_{san}, \text{ADM}), \sigma_0)$ *for* $h = (h[1], \dots, h[\ell])$*.*

PROOF. *Algorithm* Proof *on input* sk_{sig}*,* m, σ *and a sequence* (m_i, σ_i) *as well as* pk_{san} *searches the sequence to find a tuple* $(\text{TAG}_i, (j, m_i[j], pk_{sig}), r[j])$ *such that*

$$\mathsf{CHash}(pk_{san}, \text{TAG}_i, (j, m_i[j], pk_{sig}), r_i[j]) = \mathsf{CHash}(pk_{san}, \text{TAG}, (j, m[j], pk_{sig}), r[j])$$

for some distinct pair $(\text{TAG}, (j, m[j], pk_{sig}))$ *in* m, σ *and where* $\text{TAG}_i = \mathsf{PRG}(x_i)$ *for* $x_i = \mathsf{PRF}(\kappa, \text{NONCE}_i)$ *for the value* NONCE_i *in* σ_i*. If it finds such data it returns this colliding tuple together with* x_i*, i.e.,*

$$\pi = (\text{TAG}_i, (j, m_i[j], pk_{sig}), r_i[j], x_i),$$

else it outputs $\perp$*.*

JUDGE. *The judge on input* $m, \sigma, pk_{sig}, pk_{san}$ *and* $\pi = (\text{TAG}_\pi, (j, m_\pi[j], pk_{sig,\pi}), r_\pi[j], x_\pi)$ *checks that* $pk_{sig} = pk_{sig,\pi}$*, that* π *describes a non-trivial collision under* $\mathsf{CHash}(pk_{san}, \cdot, \cdot, \cdot)$ *for the pair* $((\text{TAG}, j, m[j], pk_{sig}), r[j])$ *in* σ*, i.e.,*

$$\mathsf{CHash}(pk_{san}, \text{TAG}_\pi, (j, m_\pi[j], pk_{sig,\pi}); r_\pi[j]) = \mathsf{CHash}(pk_{san}, \text{TAG}, (j, m[j], pk_{sig}); r[j]),$$

that the block j *is admissible, and that* $\text{TAG}_\pi = \mathsf{PRG}(x_\pi)$ *for the given value* x_π *in the proof. If so, it outputs* `San`*, else it returns* `Sig`*.*

Completeness of signatures generated by the signer follows easily from the completeness of the underlying signature scheme, completeness of signatures generated by the sanitizer follows from the fact that algorithm CHAdapt always returns a collision, and completeness for proofs holds as one always finds convincing data then.

5.2 Security

It remains to prove security:

Theorem 5.3 *The sanitizable signature scheme in Construction 5.2 is secure, i.e., it is immutable, transparent, sanitizer- and signer-accountable (and thus private and unforgeable), assuming that the chameleon hash function is collision-resistant under random-tagging attacks, that PRG and PRF are pseudorandom and that the signature scheme is existentially unforgeable under adaptive chosen-message attacks.*

Proof. We stepwise go through the properties. Most times we merely outline the security proof because a formalization is straightforward.

Immutability. Assume that the scheme is not immutable according to our definition and that there exists a successful adversary $\mathcal{A}$ against this property. We show that this contradicts the unforgeability of the underlying signature scheme. There are two cases: Assume that $\mathcal{A}$ succeeds by outputting $(pk^*_{\text{san}}, m^*, \sigma^*)$ such that $(pk^*_{\text{san}}, \text{ADM}^*, h^*)$ is different from all other data $(pk_{\text{san},i}, \text{ADM}_i, h_i)$ appearing in the attack. Then the valid signature σ^*_0 included in σ^* is for a message $(h^*, pk^*_{\text{san}}, \text{ADM}^*)$ which has not been signed with the underlying signature scheme before. This, however, contradicts the unforgeability of this signature scheme (observing that we can simulate Proof perfectly without knowledge of the secret key of the signature scheme).

Next assume $(pk^*_{\text{san}}, \text{ADM}^*, h^*)$ is identical to some $(pk_{\text{san},i}, \text{ADM}_i, h_i)$. Then, since $pk^*_{\text{san}} = pk_{\text{san},i}$ the messages m^* and m_i must differ in at least one inadmissible block j_i according to ADM_i. But since $\text{ADM}^* = \text{ADM}_i$ this must also be an inadmissible block according to ADM^* in m^*. Therefore $h^*[j_i] = m^*[j_i]$ must be different from $h_i[j_i] = m_i[j_i]$, contradicting the fact $h^* = h_i$. Hence, the second case cannot occur and the scheme is immutable.

Transparency. Transparency holds because with overwhelming probability all values NONCE picked by the signer are distinct and thus all x-values are computationally indistinguishable from independent and randomly chosen values. In this case all the generator's outputs, too, are indistinguishable from random $2n$-bit strings (as chosen by the sanitizer). Given this the claim now follows from the distributional property of CHAdapt, that the sanitizing process goes through all admissible block and updates them, and the fact that the distribution of the input $(h, pk_{\text{san}}, \text{ADM})$ to the signing step is independent of the message. Hence, the distribution of the reply is computationally indistinguishable in the two cases for the Sanit/Sign box, independently of further queries to the signature, sanitizing or proof oracles (using the fact that the guessing the NONCE values in the signatures computed internally in the Sanit/Sign box is infeasible).

Sanitizer-Accountability. Assume that the scheme was not sanitizer-accountable and there was a successful adversary $\mathcal{A}$, i.e., such that Proof algorithm cannot find a non-trivial collision in the chameleon hashes for $(pk^*_{\text{san}}, m^*, \sigma^*)$ and the $(pk_{\text{san},i}, m_i, \sigma_i)$ queries. First note that if $(h^*, pk^*_{\text{san}}, \text{ADM}^*) \neq (h_i, pk_{\text{san},i}, \text{ADM}_i)$

for all i, the valid signature σ_0^* in σ^* for this tuple would constitute a successful forgery against the signature scheme (using again the fact that Proof can be easily simulated without the secret signing key).

Hence, there must be some i with $(h^*, pk^*_{\text{san}}, \text{ADM}^*) = (h_i, pk_{\text{san},i}, \text{ADM}_i)$. In particular, since a success requires $(pk^*_{\text{san}}, m^*) \neq (pk_{\text{san},i}, m_i)$ we must have $m^*[j] \neq m_i[j]$ for some block j. Furthermore, because $\text{ADM}^* = \text{ADM}_i$ and inadmissible message blocks are output in clear and cannot be distinct, it holds that

$$\begin{aligned} h^*[j] &= \mathsf{CHash}(pk^*_{\text{san}}, \text{TAG}^*, (j, m^*[j], pk_{\text{sig}}); r^*[j]) \\ &= \mathsf{CHash}(pk^*_{\text{san}}, \text{TAG}_i, (j, m_i[j], pk_{\text{sig}}); r_i[j]) = h_i[j] \end{aligned}$$

for some $r^*[j]$ in σ^* and $r_i[j]$ in σ_i. This, however, implies that Proof finds such a non-trivial collision with overwhelming probability. Given this, it is clear that Proof can also output x_i from the genuine signature data.

Signer-Accountability. We finally show signer-accountability, this time using the security under random-tagging attacks of of the chameleon hash function. Assume that there is a successful attacker making the Judge accuse the sanitizer for a message which has not been sanitized by the legal sanitizer.

First note that for the adversary's successful output $pk^*_{\text{sig}}, m^*, \sigma^*$ (with tag TAG^*) and $\pi^* = (\text{TAG}_\pi, (j, m_\pi[j], pk_{\text{sig}\pi}), r_\pi[j], x_\pi)$ with overwhelming probability $\text{TAG}_\pi \neq \text{TAG}'_i$ for all i. This is so because with overwhelming probability no TAG'_i lies in the range of PRG and there cannot be a valid preimage x_π for $\text{TAG}_\pi = \text{TAG}'_i$. In particular, it follows that $\{\text{TAG}^*, \text{TAG}_\pi\} \neq \{\text{TAG}'_i, \text{TAG}'_j\}$ for all i, j.

Assume that $\{\text{TAG}^*, \text{TAG}_\pi\} \neq \{\text{TAG}_i, \text{TAG}'_i\}$ for all $i = 1, 2, \ldots, q$. Then, because we also have $\{\text{TAG}^*, \text{TAG}_\pi\} \neq \{\text{TAG}'_i, \text{TAG}'_j\}$ this would straightforwardly contradict the security of the chameleon hash (noting that we can easily simulate the sanitizer algorithm with the help of the OAdapt oracle). Hence, assume that $\{\text{TAG}^*, \text{TAG}_\pi\} = \{\text{TAG}_i, \text{TAG}'_i\}$ for some i and, since the random tags picked by the honest sanitizer are unique with overwhelming probability, we can assume that i is unique.

Because $\text{TAG}_\pi \neq \text{TAG}'_i$ we must have $\text{TAG}^* = \text{TAG}'_i$ and $\text{TAG}_\pi = \text{TAG}_i$. Since $(pk^*_{\text{sig}}, m^*) \neq (pk_{\text{sig},i}, m'_i)$ for a success there must be some j with $(\text{TAG}^*, (j, m^*[j], pk^*_{\text{sig}})) \neq (\text{TAG}'_i, (j, m'_i[j], pk_{\text{sig},i}))$. However, assuming that all sanitizer tags are unique and observing that with overwhelming probability $\text{TAG}'_i \neq \text{TAG}_i$ and that for the same tag the prepended block numbers are distinct, it follows that the adversary has generated a new collision $(\text{TAG}^*, (j, m^*[j], pk^*_{\text{sig}}))$, $(\text{TAG}'_i, (j, m_\pi[j], pk^*_{\text{sig}}))$ which has not been queried previously. This would again contradict the security of the chameleon hash function and signer-accountability follows. □

Acknowledgments

We thank the anonymous reviewers and the crypto group at Bristol for valuable comments. Marc Fischlin, Anja Lehmann and Dominique Schröder are supported by the Emmy Noether Programme Fi 940/2-1 of the German Research Foundation (DFG).

References

[ACdMT05] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable Signatures. In *ESORICS*, volume 3679 of *Lecture Notes in Computer Science*, pages 159–177. Springer, 2005.

[AdM04] Giuseppe Ateniese and Breno de Medeiros. On the Key Exposure Problem in Chameleon Hashes. In *SCN*, volume 3352 of *Lecture Notes in Computer Science*, pages 165–179. Springer, 2004.

[CLM08] Sébastien Canard, Fabien Laguillaumie, and Michel Milhau. Trapdoor Sanitizable Signatures and Their Application to Content Protection. In *ACNS*, Lecture Notes in Computer Science, pages 258–276. Springer-Verlag, 2008.

[Gol04] Oded Goldreich. *The Foundations of Cryptography*, volume 2. Cambridge University Press, 2004.

[HHH+08] Stuart Haber, Yasuo Hatano, Yoshinori Honda, William Horne, Kunihiko Miyazaki, Tomas Sander, Satoru Tezoku, and Danfeng Yao. Efficient signature schemes supporting redaction, pseudonymization, and data deidentification. In *ASIACCS*, pages 353–362. ACM, 2008.

[IKO+07] Tetsuya Izu, Noboru Kunihiro, Kazuo Ohta, Masahiko Takenaka, and Takashi Yoshioka. A Sanitizable Signature Scheme with Aggregation. In *ISPEC*, volume 4464 of *Lecture Notes in Computer Science*, pages 51–64. Springer, 2007.

[IKTY05] Tetsuya Izu, Nobuyuki Kanaya, Masahiko Takenaka, and Takashi Yoshioka. PIATS: A Partially Sanitizable Signature Scheme. In *ICICS*, volume 3783 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2005.

[JMSW02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic Signature Schemes. In *CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 244–262. Springer, 2002.

[KL06] Marek Klonowski and Anna Lauks. Extended Sanitizable Signatures. In *ICISC*, volume 4296 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2006.

[MHI08] Kunihiko Miyazaki, Goichiro Hanaoka, and Hideki Imai. Invisibly Sanitizable Digital Signature Scheme. *IEICE Transactions*, 91-A(1):392–402, 2008.

[MIM+05] Kunihiko Miyazaki, Mitsuru Iwamura, Tsutomu Matsumoto, Ryôichi Sasaki, Hiroshi Yoshiura, Satoru Tezuka, and Hideki Imai. Digitally Signed Document Sanitizing Scheme with Disclosure Condition Control. *IEICE Transactions*, 88-A(1):239–246, 2005.

[MSI+03] K. Miyazaki, S. Susaki, M. Iwamura, T. Matsumoto, R. Sasaki, and H. Yoshiura. Digital documents sanitizing problem. In *Technical Report ISEC2003-20*. IEICE, 2003.

[SBZ01] Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content Extraction Signatures. In *ICISC*, volume 2288 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2001.

[SIT06] Manabu Suzuki, Toshiyuki Isshiki, and Keisuke Tanaka. Sanitizable signature with secret information. In *In Proceedings of the Symposium on Cryptography and Information Security*. ACM, 2006.

[YSLM08] Tsz Hon Yuen, Willy Susilo, Joseph K. Liu, and Yi Mu. Sanitizable Signatures Revisited. In *CANS*, volume 5339 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 2008.

A Semantic Privacy

We formalize semantic privacy similar to the definition of semantic security of encryption schemes for multiple messages. We do so following the approach of a posteriori CCA security due to Goldreich [Gol04]. Note that this definition is also equivalent to the definition of indistinguishability of encryption schemes.

Roughly speaking, semantic privacy means that everything that can be learned about the original message from a sequence of signatures, can also be learned without these signature, i.e., from the sanitized messages only. This intuition is formalized through two experiments where, in one case, an adversary $\mathcal{A}$ can (adaptively) submit descriptions $\mathcal{M}_1, \ldots, \mathcal{M}_q$ of distributions to a tester oracle $\mathsf{T}_{\mathcal{A}}$ which samples messages $m_1, \ldots, m_q$ and modifications according to the distributions and returns the sanitized messages $m'_1, \ldots, m'_q$ and signatures for them.

In the other experiment a simulator $\mathcal{S}$ may also submit distributions $\mathcal{M}_1, \ldots, \mathcal{M}_q$ to an oracle $\mathsf{T}_{\mathcal{S}}$ which samples messages too, but this time only returns the sanitized messages $m'_1, \ldots, m'_q$ to the simulator (without

the signatures). The goal of both parties is to predict some information $f(m_1, \ldots, m_q, \mathcal{M}_1, \ldots, \mathcal{M}_q)$ about the original messages, and the security requirement is now that both parties should be (almost) equally successful. Note that we include the chosen distributions $\mathcal{M}_1, \ldots, \mathcal{M}_q$ into the probabilistic function f to prevent trivial choices of the simulator.

More formally, the tester oracle $\mathsf{T}_{\mathcal{A}}$ in $\mathcal{A}$'s experiment takes as input a distribution $\mathcal{M}$ and two public/secret key pairs $(sk_{\text{sig}}, pk_{\text{sig}}), (sk_{\text{san}}, pk_{\text{san}})$. It first samples $(m, \text{MOD}, \text{ADM}) \leftarrow \mathcal{M}$ and computes the signature $\sigma \leftarrow \mathsf{Sign}(m, sk_{\text{sig}}, pk_{\text{san}}, \text{ADM})$, as well as the sanitized message/signature pair $(m', \sigma') \leftarrow \mathsf{Sanit}(m, \text{MOD}, \sigma, pk_{\text{sig}}, sk_{\text{san}})$. It returns the pair $(m', \text{MOD}, \text{ADM}, \sigma')$. The tester oracle $\mathsf{T}_{\mathcal{S}}$ for the simulator $\mathcal{S}$ is identical to $\mathsf{T}_{\mathcal{A}}$ except that $\mathsf{T}_{\mathcal{S}}$ returns only the triple $(m', \text{MOD}, \text{ADM})$. With these description of the oracles we obtain:

Definition A.1 *A sanitizable signature scheme* SanSig *is* semantically private*, if for any efficient algorithm $\mathcal{A}$, there exists an efficient algorithm $\mathcal{S}$ such that for any efficient probabilistic function f the difference*

$$\left|\text{Prob}\left[\mathsf{SemPriv}_{\mathcal{A}}^{\mathsf{SanSig\text{-}0}}(n) = 1\right] - \text{Prob}\left[\mathsf{SemPriv}_{\mathcal{S}}^{\mathsf{SanSig\text{-}1}}(n) = 1\right]\right| \approx 0\,,$$

is negligible, where the experiments are defined as follows:

Experiment $\mathsf{SemPriv}_{\mathcal{A}}^{\mathsf{SanSig\text{-}0}}(n)$
$(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^n)$
$(pk_{san}, sk_{san}) \leftarrow \mathsf{KGen}_{san}(1^n)$
$v \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{sig}, pk_{san}, \cdot), \mathsf{Sanit}(\cdot,\cdot,\cdot, pk_{sig}, sk_{san}), \mathsf{Proof}(sk_{sig}, \cdots), \mathsf{T}_{\mathcal{A}}(sk_{sig}, pk_{sig}, sk_{san}, pk_{san}, \cdot)}(pk_{sig}, pk_{san})$
let m_i and $\mathcal{M}_i$ for $i = 1, 2, \ldots, q$ denote the messages m_i sampled according to the distributions $\mathcal{M}_i$ submitted to $\mathsf{T}_{\mathcal{A}}$.
return 1 *iff* $f(m_1, \ldots, m_q, \mathcal{M}_1, \ldots, \mathcal{M}_q) = v$*, else* 0*.*

and

Experiment $\mathsf{SemPriv}_{\mathcal{S}}^{\mathsf{SanSig\text{-}1}}(n)$
$(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^n)$
$(pk_{san}, sk_{san}) \leftarrow \mathsf{KGen}_{san}(1^n)$
$v \leftarrow \mathcal{S}^{\mathsf{Sign}(\cdot, sk_{sig}, pk_{san}, \cdot), \mathsf{Sanit}(\cdot,\cdot,\cdot, pk_{sig}, sk_{san}), \mathsf{Proof}(sk_{sig}, \cdots), \mathsf{T}_{\mathcal{S}}(sk_{sig}, pk_{sig}, sk_{san}, pk_{san}, \cdot)}(pk_{sig}, pk_{san})$
let m_i and $\mathcal{M}_i$ for $i = 1, 2, \ldots, q$ denote the messages m_i sampled according to the distributions $\mathcal{M}_i$ submitted to $\mathsf{T}_{\mathcal{S}}$.
return 1 *iff* $f(m_1, \ldots, m_q, \mathcal{M}_1, \ldots, \mathcal{M}_q) = v$*, else* 0*.*

We prove that the definitions of semantically private sanitizable signature schemes and private schemes are equivalent.

Theorem A.2 *A sanitizable signature scheme is semantically private if and only if it is private.*

We prove Theorem A.2 with the following to propositions.

Proposition A.3 *Every private sanitizable signature scheme is also semantically private.*

Proof. Suppose that SanSig is a private sanitizable signature scheme. We show that SanSig is a semantically private sanitizable signature scheme by constructing for every efficient adversary $\mathcal{A}$ an efficient simulator $\mathcal{S}$. The construction follows the proof ideas of encryption schemes, namely, building a simulator which executes a black-box simulation of $\mathcal{A}$ on random values and showing that the indistinguishability property of privacy implies that the success probability of $\mathcal{A}$ and $\mathcal{S}$ is nearly the same.

Construction of $\mathcal{S}$. Let $\mathcal{A}$ be an efficient algorithm which deduces some partial information, i.e., the value $f(m_1, \ldots, m_q, \mathcal{M}_1, \ldots, \mathcal{M}_q)$. The simulator $\mathcal{S}$ gets as input two public keys $(pk_{\text{sig}}, pk_{\text{san}})$, it executes $\mathcal{A}$ on these keys and simulates the signing oracle, the sanitizer oracle, and the proof oracle with its own oracles. Whenever $\mathcal{A}$ queries its tester oracle $\mathsf{T}_{\mathcal{A}}$ on a distribution $\mathcal{M}$, then $\mathcal{S}$ invokes $\mathsf{T}_{\mathcal{S}}$ on $\mathcal{M}$ and gets a triple $(m', \text{MOD}, \text{ADM})$. Algorithm $\mathcal{S}$ picks a message m at random such that applying MOD to m yields m' and computes the corresponding signature with the help of its signing oracle $\sigma \leftarrow \mathsf{Sign}(\cdot, sk_{\text{sig}}, pk_{\text{san}}, \cdot)$. In order to obtain a sanitized signature, $\mathcal{S}$ queries the sanitize oracle $\sigma' \leftarrow \mathsf{Sanit}(\cdot, \cdot, \cdot, pk_{\text{sig}}, sk_{\text{san}})$ about (m', MOD, σ). It returns the tuple $(m', \text{MOD}, \text{ADM}, \sigma')$. At the end of the simulation, $\mathcal{A}$ stops, eventually outputting a value v which $\mathcal{S}$ also outputs.

Analysis. For the analysis note that $\mathcal{S}$ is efficient because $\mathcal{A}$ is efficient and since the overhead of handling $\mathcal{A}$'s queries can be done in polynomial time. It is still left to show that the success probability of $\mathcal{A}$ and of $\mathcal{S}$ (as described below) are nearly the same:

$$\left|\text{Prob}\left[\mathsf{SemPriv}_{\mathcal{A}}^{\text{SanSig-0}}(n) = 1\right] - \text{Prob}\left[\mathsf{SemPriv}_{\mathcal{S}}^{\text{SanSig-1}}(n) = 1\right]\right| = \epsilon(n) \approx 0\,.$$

Intuitively, the equation follows from privacy. Let us assume towards contradiction that $\epsilon(n)$ is noticeable. Then we show how to build an algorithm $\mathcal{B}$ breaking privacy as follows.

Algorithm $\mathcal{B}$ gets as input the public key of the signer pk_{sig} as well as the public key of the sanitizer pk_{san}. It runs a black-box simulation of $\mathcal{A}$ on input $(pk_{\text{sig}}, pk_{\text{san}})$ and answers each signing and sanitizing oracle query with its own oracles. During the simulation, $\mathcal{A}$ may query its tester oracle $\mathsf{T}_{\mathcal{A}}$ on some distribution $\mathcal{M}$ which algorithm $\mathcal{B}$ handles as follows. It picks randomly a triple $(m_0, \text{MOD}_0, \text{ADM}_0) \leftarrow \mathcal{M}$, and selects another random message m_1 such that both messages m_0, m_1 map to the same *modified* message m', i.e., $(m_0, \text{MOD}_0, \text{ADM}_0) \equiv (m_1, \text{MOD}_1, \text{ADM}_1)$. It invokes its external $\mathsf{LoRSanit}$ oracle with theses pairs, receives (m'_b, σ'_b) and forwards this pair to $\mathcal{A}$. Algorithm $\mathcal{B}$ stores in each execution the message $m_0^{(i)}$ as well as the queried distribution $\mathcal{M}_i$ and let $q(n)$ be the number of oracle invocations. When $\mathcal{A}$ stops, eventually outputting a value v, attacker $\mathcal{B}$ returns 0 iff $f\left(m_0^{(1)}, \ldots, m_0^{(q)}, \mathcal{M}_1, \ldots, \mathcal{M}_q\right) = v$, otherwise 1.

For the analysis of $\mathcal{B}$ note that $\mathcal{B}$ is efficient since $\mathcal{A}$ runs in polynomial-time and because the overhead of choosing the messages and handling $\mathcal{A}$ queries can all be done efficiently. Let us take a look at the choice of the bit b in the indistinguishability experiment and its effect on our simulation. If the left-or-right oracle is initialized with $b = 0$, then $\mathcal{A}$ receives the sanitized message-signature pair (m', σ') on message m_0 which was previously chosen by $\mathcal{A}$ and the second message is ineffectual. This behavior corresponds exactly to the tester oracle $\mathsf{T}_{\mathcal{A}}$, because all values are randomly chosen. In the second case, where the bit b equals 1 in the indistinguishability experiment, $\mathcal{A}$ receives the sanitized message-signature pair (m', σ') derived from the randomly chosen message m_1. This, however, reflects the behavior of the second tester oracle $\mathsf{T}_{\mathcal{S}}$.

Next, we have to analyze the success probability of $\mathcal{B}$ in predicting the right bit $b^* = b$:

$$\begin{aligned}&\text{Prob}\left[\mathsf{Privacy}_{\mathcal{B}}^{\text{SanSig}}(n) = 1\right]\\ &\quad = \tfrac{1}{2}\left(\text{Prob}\left[\mathsf{Privacy}_{\mathcal{B}}^{\text{SanSig}}(n) = 1 \;\middle|\; b = 0\right] - \text{Prob}\left[\mathsf{Privacy}_{\mathcal{B}}^{\text{SanSig}}(n) = 1 \;\middle|\; b = 1\right]\right) + \tfrac{1}{2}\end{aligned}$$

Note that our construction maps the choice of the bit b to the different types of experiments in the semantic security experiment, thus

$$= \tfrac{1}{2}\left(\text{Prob}\left[\mathsf{SemPriv}_{\mathcal{A}}^{\text{SanSig-0}}(n) = 1\right] - \text{Prob}\left[\mathsf{SemPriv}_{\mathcal{S}}^{\text{SanSig-1}}(n) = 1\right]\right) + \tfrac{1}{2}\,.$$

By our assumption that the difference is noticeable, it follows that the overall probability of $\mathcal{B}$ is noticeably greater than $\frac{1}{2}$. This, however, contradicts the privacy. □

Proposition A.4 *Every semantically private sanitizable signature scheme is also private.*

Proof. Before proving the proposition we note that it suffices to consider the case where the adversary queries its left-or-right oracle only once. Applying a standard hybrid argument yields a proof for the case of multiple-message queries.

Let $\mathcal{A}$ be an adversary in the privacy experiment. This adversary invokes its external oracle LoRSanit only once with the tuples (m_0, MOD_0) and (m_1, MOD_1) for the same ADM. We then define an adversary $\mathcal{A}'$ which is identical to $\mathcal{A}$ but which is executed in the semantic privacy experiment. When adversary $\mathcal{A}$ submits the pair $(m_0, \text{MOD}_0, \text{ADM}_0)$, $(m_1, \text{MOD}_1, \text{ADM}_1)$ to its left-or-right oracle we have $\mathcal{A}'$ prepare the distribution $\mathcal{M}$ by the canonical circuit assigning each tuple $(m_0, \text{MOD}_0, \text{ADM}_0)$, $(m_1, \text{MOD}_1, \text{ADM}_1)$ the same probability $1/2$. Then $\mathcal{A}'$ submits this distribution to its tester oracle and forwards the answer to $\mathcal{A}$. When $\mathcal{A}$ finally outputs a guess b^* we let $\mathcal{A}'$ output this value, too.

We consider some special information f about the original message m_b in the experiment of $\mathcal{A}$ and of $\mathcal{A}'$, respectively. The function f first of all excludes any distribution $\mathcal{M}$ which is not of the above type. That is, call a distribution *trivial* if it does not constitute a canonical description of a distribution over two tuples, each tuple appearing with the same probability $1/2$, or if applying MOD to both messages m_0 and m_1 does not yield the same message m', or if $m_0 = m_1$, or if the admissible blocks are distinct $\text{ADM}_0 \neq \text{ADM}_1$. Note that m_0 and m_1 are recoverable from the description of $\mathcal{M}$. Define f now by

$$f(m_b, \mathcal{M}) = \begin{cases} b' & \text{for random bit } b' \text{ if } \mathcal{M} \text{ is trivial} \\ 0 & \text{if } m_b = m_0 \ \mathcal{M} \text{ is not trivial} \\ 1 & \text{if } m_b = m_1 \ \mathcal{M} \text{ is not trivial.} \end{cases}$$

Note that, with the specification of f, adversary $\mathcal{A}'$ wins in the semantic privacy experiment with the same probability as $\mathcal{A}$ in the regular privacy experiment. But then there exists a simulator $\mathcal{S}$ which succeeds with the same probability as $\mathcal{A}'$ for function f (except for a negligible amount). For any simulator the probability of winning its experiment is exactly $1/2$, though, because for trivial distributions $\mathcal{M}$ submitted by the simulator this is clear and for non-trivial distributions it follows as the sanitized message does not leak any information about the starting message. Thus, the probability that $\mathcal{A}$ predicts the right bit b in the regular privacy experiment is negligibly close to $1/2$. □

B General Message Modifications

In this section we outline how to adapt our security notions for more general message modifications. To this end we assume that ADM and MOD are (descriptions of) efficient algorithms such that $\text{ADM}(\text{MOD}) \in \{0, 1\}$ indicates if the modification is admissible and matches ADM, i.e., $\text{ADM}(\text{MOD}) = 1$. The function MOD maps any message m to the modified message $m' = \text{MOD}(m)$.

The notion of unforgeability remains unchanged. For immutability we demand as before that the adversary's output $(pk^*_{\text{san}}, m^*, \sigma^*)$ describes a valid message-signature pair under keys $pk_{\text{sig}}, pk^*_{\text{san}}$. With the general message modification we now require for all queries to the signing oracle for $i = 1, 2, \ldots, q$ that $pk^*_{\text{san}} \neq pk_{\text{san},i}$ or $m^* \notin \{\text{MOD}(m_i) \mid \text{MOD with } \text{ADM}(\text{MOD}) = 1\}$. Note that, under this general definition, it may not be efficiently verifiable if the adversary has succeeded.

The notion of privacy under general modifications demands that for each pair $(m_{j,0}, \text{MOD}_{j,0}, \text{ADM}_j)$, $(m_{j,1}, \text{MOD}_{j,1}, \text{ADM}_j)$ submitted to the left-or-right oracle the modifications are admissible and yield the same message, i.e., we simply adapt the notation $(m_{j,0}, \text{MOD}_{j,0}, \text{ADM}_j) \equiv (m_{j,1}, \text{MOD}_{j,1}, \text{ADM}_j)$ accordingly. Transparency and the accountability notions remain unchanged.

We note that both security implications (transparency implies privacy and accountability implies unforgeability) are also valid under this more general notion. The separations remain true as block-based descriptions of MOD and ADM constitute a special case.

C Relationship of Unforgeability and Accountability Revisited

Recall that we have shown that sanitizer- and signer-accountability together imply unforgeability. Here we show that neither accountability notion alone is sufficient (even if the other properties like transparency hold).

Proposition C.1 (Independence of Unforgeability and Sanitizer-Accountability) *Assume that there exists a secure sanitizable signature scheme. Then there exists a sanitizable signature scheme which is immutable, transparent and signer-accountable, but not unforgeable nor sanitizer-accountable.*

Proof. As in the separation results in Section 4 let $\mathsf{SanSig} = (\mathsf{KGen}_{\mathrm{sig}}, \mathsf{KGen}_{\mathrm{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ be once more a secure sanitizable signature scheme and let $\mathsf{SanSig}' = (\mathsf{KGen}_{\mathrm{sig}}, \mathsf{KGen}'_{\mathrm{san}}, \mathsf{Sign}', \mathsf{Sanit}', \mathsf{Verify}', \mathsf{Proof}', \mathsf{Judge}')$ be the following modification of SanSig:

- $\mathsf{KGen}_{\mathrm{sig}}$ remains unchanged.
- $\mathsf{KGen}'_{\mathrm{san}}$ works as $\mathsf{KGen}_{\mathrm{san}}$, except that it concatenates a '1' to the generated public key, $pk'_{\mathrm{san}} = pk_{\mathrm{san}}||1$.
- algorithms Sign' and Sanit' work as the originals except that, if they have produced a signature for a single-block message which is also modifiable, then they append an additional signature σ_0 for the modifiable all-zero message $m_0 = 0^t$ for the same public keys to the signature (Sign' can do this from scratch and Sanit' can simply modify the given message to m_0 and derive the signature).
- Judge' takes the public key of the sanitizer and outputs `Sig` if this public key ends with '0'. Else, it runs Judge on the input data (and the public key pk'_{san} with the last bit dropped).
- all other algorithms merely chop off the last bit of pk'_{san} and work as their ancestors (but using the shortened key).

The scheme is clearly not unforgeable. An adversary can easily query the signature oracle about a single-block message $m \neq 0^t$ (which is also mutable) to recover an extra signature σ_0 for m_0, and then output this pair. By the completeness of the underlying scheme Verify' returns `true` for this pair and the adversary has won. The scheme is neither sanitizer-accountable because an attacker can simply choose a key pk^*_{san} ending with '0', use the same trick as in the unforgeabilty case to derive an extra signature σ_0 for m_0 and force the judge to output `Sig` for this new and valid message-signature pair and the key ending with '0'.

It remains to show that the other properties are preserved. Completeness is obviously inherited. As for immutability note that a query yielding the extra signature σ_0 for m_0 makes single-block messages useless for a successful attack because m_0 is a trivial derivation of the query. More formally, given a successful immutability adversary against the new scheme SanSig' one could easily construct a successful immutability adversary against the underlying scheme, simply by making an additional call to the signing oracle about m_0 in case of queries about single-block, modifiable messages.

Regarding transparency any attacker winning in the modified scheme could be turned into a successful transparency adversary in the original scheme. The derived adversary would only need to make an additional call to the $\mathsf{Sanit}/\mathsf{Sign}$ box about m_0 whenever the adversary for the modified scheme asks this box about a single-block, mutable message. Finally, signer-accountability is trivially true as the new judge for honestly generated keys pk'_{san} always runs in the regular mode. □

Proposition C.2 (Independence of Unforgeability and Signer-Accountability) *Assume that there exists a secure sanitizable signature scheme. Then there exists a sanitizable signature scheme which is immutable, transparent and sanitizer-accountable, but not unforgeable nor signer-accountable.*

Proof. Let again $\mathsf{SanSig} = (\mathsf{KGen}_{\mathrm{sig}}, \mathsf{KGen}_{\mathrm{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ be a secure sanitizable signature scheme and consider the modifed scheme $\mathsf{SanSig}' = (\mathsf{KGen}'_{\mathrm{sig}}, \mathsf{KGen}_{\mathrm{san}}, \mathsf{Sign}', \mathsf{Sanit}', \mathsf{Verify}', \mathsf{Proof}', \mathsf{Judge}')$:

- $\mathsf{KGen}'_{\text{sig}}$ works as $\mathsf{KGen}_{\text{sig}}$, except that it concatenates a '1' to the generated public key, $pk'_{\text{sig}} = pk_{\text{sig}}||1$.
- $\mathsf{KGen}_{\text{san}}$ remains unchanged.
- algorithms Sign' and Sanit' work as the originals except that, if they have produced a signature for a single-block message which is also modifiable, then they append an additional signature σ_0 for the modifiable all-zero message $m_0 = 0^t$ for the same public keys to the signature (Sign' can do this from scratch and Sanit' can simply modify the given message to m_0 and derive the signature).
- Judge' takes the public key of the sanitizer and outputs $\mathtt{San}$ if this public key ends with '0'. Else, it runs Judge on the input data (and the public key pk'_{san} withthe last bit dropped).
- all other algorithms merely chop off the last bit of pk'_{san} and work as their ancestors (but using the shortened key).

As in the previous proof it follows that the scheme is not unforgeable and not signer-accountable. Completeness is preserved, and immutability and transparency follow as before. Analogously, sanitizer-accountability is preserved as Judge' never enters the exceptional mode in this case. □

D Chameleon Hashing Secure Against Random-Tagging Attacks

In this section we construct a chameleon hash function which provides security under random-tagging attacks, relying on the RSA assumption and the random oracle model. The idea is to use a standard RSA-based chameleon hash function $(m, r) \mapsto g^m r^e \bmod N$, but generate g from Tag and m via the random oracle H. As m already enters the computation for $g = H(\text{Tag}, m)$ this makes the explicit exponentiation with m obsolete and the hash value becomes $gr^e \bmod N$.

Construction D.1 *Define the following chameleon hash function $\mathcal{CH} = (\mathsf{CHKGen}, \mathsf{CHash}, \mathsf{CHAdapt})$ in the presence of a random oracle $H : \{0,1\}^* \to \mathbb{Z}_N^*$:*

Key Generation. *Algorithm CHKGen on input 1^n generates RSA parameters N, e (with e being prime) and d for parameter 1^n. It returns $pk = (N, e)$ and $sk = (N, d)$.*

Hashing. *Algorithm CHash on input pk, Tag, m and randomness r computes $g = H(\text{Tag}, m)$ and returns $y = gr^e \bmod N$.*

Adapting. *Algorithm $\mathsf{CHAdapt}$ in input sk, Tag, m, r and Tag$'$, m' computes $g = H(\text{Tag}, m)$, $y = gr^e \bmod N$ and $g' = H(\text{Tag}', m')$ and returns Tag$'$ together with $r' = (y(g')^{-1})^d \bmod N$.*

A simple calculation confirms that $\mathsf{CHAdapt}$ returns a collision for the given data, and that the value is randomly distributed in $\mathbb{Z}_N^*$ (if r is). It remains to show collision-resistance:

Proposition D.2 *The chameleon hash function in Construction D.1 is collision-resistant under random-tagging attacks in the random oracle model under the RSA assumption.*

Proof. Assume towards contradiction that the hash function was not secure and that there exists a successful adversary $\mathcal{A}$ in experiment $\mathsf{RndTag}_{\mathcal{A}}^{\mathcal{CH}}(n)$. We then show how to find for given (N, e, z) the value $z^{1/e} \bmod N$ via black-box simulation of $\mathcal{A}$. To this end, we first describe a simulating strategy for algorithm $\mathsf{CHAdapt}$ without knowledge of the secret exponent d, but by programming the random oracle:

random oracle. Whenever the adversary queries the random oracle H about a (new) value, we return $z^x t^e \bmod N$ for random $x \leftarrow \mathbb{Z}_e$ and $t \leftarrow \mathbb{Z}_N^*$. As usual, if the query has been made before, we simply return the same answer as before again.

ORACLE OADAPT. When the adversary submits a query (TAG, m, r, m') to oracle OAdapt we first pick TAG' at random. If this value has appeared before we stop outputting "failure", else we compute $g = H(\text{TAG}, m)$ (possibly querying the simulated random oracle) and $y = gr^e \bmod N$. We set the random oracle value $H(\text{TAG}', m')$ to be $g' = y(r')^{-e} \bmod N$ for random $r' \leftarrow \mathbb{Z}_N^*$ and return (TAG', r').

Note that, given that we do not stop prematurely because of a colliding value TAG', the simulation is perfect from $\mathcal{A}$'s point of view. In particular, the simulation of oracle OAdapt yields $g'(r')^e = y \bmod N$ and g' is a random group element (and r' is uniquely deterimened by y, g' and N, e).

Now assume that the adversary wins in the simulation and outputs $(\text{TAG}, m, r) \neq (\text{TAG}', m', r')$. Assume that the adversary has also made the corresponding queries for this final output to the simulated random oracle; else make these queries now. By definition not both values (TAG, m), (TAG', m') could have appeared in the answers of the simulated oracle OAdapt for a successful attack. Hence, one of them, say (TAG, m), yields a value $g = H(\text{TAG}, m)$ of the form $g = z^x t^e \bmod N$ via a simulated random oracle call.

Note that the second value (TAG', m'), too, must yield an element $g' = H(\text{TAG}', m')$ of the form $g' = z^{x'}(t')^e \bmod N$, either through a direct query to the simulated random oracle or by setting g' via the simulated collision-finder. In the first case, x' is clearly independent of x as $(\text{TAG}, m) \neq (\text{TAG}', m')$. In the second case, since a success requires $\{(\text{TAG}, m), (\text{TAG}', m')\}$ to be distinct from the pairs appearing in each communication with oracle OAdapt, the reply including TAG' must have been for a different query $(\text{TAG}_i, m_i) \neq (\text{TAG}, m)$. In other words, $g = H(\text{TAG}_i, m_i)$ is of the form $g = z^{x'}(t'')^e \bmod N$ for an x', independent of x, and so is $g' = g(r(r')^{-1})^e \bmod N$.

In any case we obtain a collision $z^x(tr)^e = z^{x'}(t'r')^e \bmod N$ for independent $x, x' \in \mathbb{Z}_e$. Hence, except with negligible probability $1/e$ with which $x = x' \bmod e$, we obtain distinct representations, from which computing an e-th root of z is easy. □