Sanitizable Signcryption: Sanitization over Encrypted Data (Full Version)

Victoria Fehr[1] Marc Fischlin[1]

Cryptoplexity, Technische Universität Darmstadt, Germany
www.cryptoplexity.de
victoria.fehr@cased.de marc.fischlin@cryptoplexity.de

Abstract.

We initiate the study of sanitizable signatures over encrypted data. While previous solutions for sanitizable signatures require the sanitizer to know, in clear, the original message-signature pair in order to generate the new signature, we investigate the case where these data should be hidden from the sanitizer and how this can be achieved with encryption. We call this primitive sanitizable signcryption, and argue that there are two options concerning what the sanitizer learns about the sanitized output: in semi-oblivious sanitizable signcryption schemes the sanitizer may get to know the sanitized message-signature pair, while fully oblivious sanitizable signcryption schemes even protect the output data. Depending on the application, either notion may be preferable.

We continue to show that semi-oblivious sanitizable signcryption schemes can be constructed in principle, using the power of multi-input functional encryption. To this end, we wrap a regular sanitizable signature scheme into a multi-input functional encryption scheme, such that functional decryption corresponds to the sanitization process. Remarkably, the multi-input functional encryption scheme cannot easily be transferred to a fully oblivious sanitizable signcryption version, so we give a restricted solution based on fully homomorphic encryption for this case.

1 Introduction

Sanitizable signatures allow a designated party, the sanitizer, to alter parts of a signed message and to derive a signature for the new message. Here, the original signer determines the admissible modifications during signature creation and the scheme enforces that the sanitizer cannot perform other operations to get a valid signatures. Introduced by Ateniese et al. [ACdT05] and, in a slightly different version, by Steinfeld et al. [SBZ02] and Miyazaki et al. [MSI+03], sanitizable signatures are related to concepts like redactable [SBZ02, JMSW02, HHH+08] and homomorphic [JMSW02, ABC+12] signatures but differ in an important aspect from these primitives: sanitization is linked to the secret sanitization key, whereas redactable and homomorphic schemes support only public modifications.

1.1 Sanitization over Encrypted Data

Usually, sanitizable signatures are not concerned about the confidentiality of the message towards the sanitizer, but only about the sanitized parts towards receivers. That is, the sanitizer becomes aware of the original message and only after sanitization the modified parts hide the original data. In fact, many sanitizable signature schemes actually require the sanitizer to use the original message to create the new message and signature, e.g., [ACdT05, CLM08, BFF+09, BFLS10]. In other words, the signer needs to trust the sanitizer not to reveal the original data, deliberately or unintentionally.

A straightforward idea to reduce the trust required from the sanitizer is to hide the original data via encryption. The setting somehow advocates a "sign-then-encrypt" approach, instead of the common "encrypt-then-sign" to secure communication, because, ultimately, in a sanitizable signature scheme the receiver should retrieve the signature over the sanitized message in clear.[1] Still, in order not to exclude any options to build solutions, we more broadly speak of sanitizable signcryption schemes, allowing either of the orders of using signatures with encryption, or even more advanced combinations.

There are two "natural" options to capture the desired formalism for sanitizable signcryption. We motivate them through corresponding application examples. The first setting is the one in which the sanitizer remains unaware of the original message but learns the sanitized message. We call this *semi-oblivious sanitizable signcryption* and the setting is depicted in Figure 1. A typical application may be electronic health records where only administrative data, say, about procedures, may appear in the sanitized message, but any clinical information like diagnoses should also be omitted from the sanitizer. Here, the sanitizer may be generally responsible for such administrative data and, therefore, has good reason to learn the sanitized message.

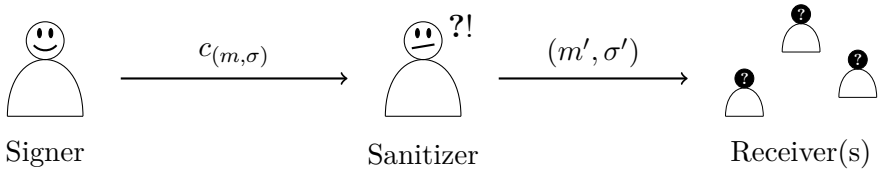

Figure 1: Pictorial overview of semi-oblivious sanitizable signcryption: the sanitizer does not learn the original cleartexts but sees the sanitized data after processing.

[1] An extra layer of complication for constructions following the encrypt-then-sign approach would also be that the signer would need to be able to translate admissible operations on the message to admissible operations on the ciphertext of the message.

The other option is to use *fully oblivious sanitizable signcryption* schemes (or, for short, oblivious sanitizable signcryption schemes) where the sanitizer basically only sees an encrypted version of the sanitized data, as shown in Figure 2. Note that this case appears to be much more challenging since the sanitizer has to perform the entire sanitization task obliviously, and that we somehow need to make sure that the receiver can decrypt the sanitized data. A typical application example may be a secure transmission from the signer to the receiver where the ISP, acting as a sanitizer here, may prune data if the receiver indicates a current lack of bandwidth (e.g., transmitting only textual parts of e-mails).[2]

Both models inherit the basic security properties of sanitizable signatures: unforgeability, immutability (preventing inadmissible modifications of the malicious sanitizer), privacy (hiding the sanitized parts), unlinkability (guaranteeing that sanitized signatures of the same origin cannot be linked), and accountability (allowing a judge to distinguish signatures originating from the signer or sanitizer). In addition, we introduce the notion of indistinguishability, akin to chosen-ciphertext security of encryption schemes, which prevents any party apart from the signer, thus especially the sanitizer, to learn the original data for semi-oblivious sanitizable signcryption schemes. For the fully oblivious sanitizable signcryption case, one additionally requires that only the (designated) receiver learns the sanitized message-signature pair.

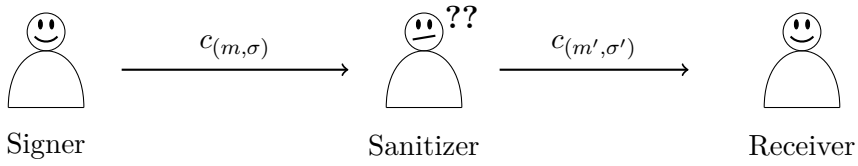

Figure 2: Pictorial overview of fully oblivious sanitizable signcryption: the sanitizer here does not even learn the sanitized data after processing.

1.2 Constructing Sanitizable Signcryption Schemes

We present two constructions showing the feasibility of building sanitizable signcryption schemes. The first one for semi-oblivious sanitizable signcryption schemes is based on a combination of a regular sanitizable signatures with multi-input functional encryption [GGG+14]. Multi-input functional encryption (miFE) can be viewed as a generalization of functional encryption [SW08, BSW11, O'N10] where applying a decryption key sk_f to a ciphertext results in the multi-input function f of the underlying inputs. The initial proposal of miFE contains a construction based on indistinguishability obfuscation. Later, several other constructions based on weaker assumptions, offering various trade-offs between efficiency and security, were proposed [BLR+14, BKS15, AJ15].

The idea is now to let the signer use the regular sanitizable encryption scheme to create the original signature and link the admissible changes to the sanitizer's signature key. Then, encapsulate these data in a multi-input functional encryption scheme, where the sanitizer holds the decryption key sk_f for the function f which performs exactly the sanitization process on the data, including potential checks for admissibility of operations. The security of the miFE scheme implements the confidentiality of the original data, and the other required security properties of sanitizable signcryption schemes are guaranteed through the inner sanitizable scheme. Interestingly, the non-triviality requirement of miFE schemes about function

[2]This, sadly, was also Nokia's intention when they decrypted, through a man-in-the-middle kind of process, HTTPS-encrypted data run through the Xpress browsers and routed through their servers, as reported in 2013. Their goal was to compress traffic to save on the receiver's side. Their solution broke end-to-end security, even without letting the users be aware of the risk. See [Mey13] for more details. Note that we do not claim that sanitizable signcryption offhandedly solves that problem, but it is a first step towards balancing out confidentiality requirements with the ability to modify data in a controlled way.

inputs, called compatibility, imposes on us to start with a perfectly private sanitizable scheme, such as in [BFLS10].

To construct fully oblivious sanitizable signcryption schemes it seems now straightforward to augment the above solution, but this time letting the function f not only sanitize, but finally also re-encrypt the sanitized pair under the receiver's public key. This, however, does not work, exactly because of the compatibility requirement, as we will discuss. We therefore settle for a weaker solution based on fully homomorphic encryption (FHE) [Gen09], where the sanitizer operates on the encrypted data, executing the underlying sanitizable scheme "through the encryption", and the output remains encrypted.

The FHE solution comes with some caveats, though. Firstly, since FHE schemes do not provide full chosen-ciphertext security we settle for a weaker, chosen-plaintext-like solution for our setting. Secondly, we do not obtain a scheme which is publicly verifiable in the sense that anyone can check the validity of the sanitized version. The reason is that, even if our homomorphic scheme is able to evaluate the verification operation in principle, the result is an encryption of the decision bit. Only the designated receiver can verify (and possibly publish later) the sanitized message-signature pair. Due to the structure of FHE schemes, compared to the functional encryption case, the signer needs to know the intended receiver in advance. In some cases, it may be easier then for the signer to sanitize and re-sign the message directly. However, the example with the ISP displays a case where the receiver is known beforehand, and the decision whether sanitization should be applied or not, can be made by the sanitizer "on the fly". This is still possible with our FHE-based construction. Overcoming any, or even all of these restrictions remains an open problem.

1.3 Related Work

Sanitizable signature schemes meanwhile come in various flavors in terms of multiple users, with n signers and m sanitizers [CJL12], or of extra functionality. For instance, identity-based solutions have been considered in [MSP10], aggregation has been added in [IKO+07], trapdoor sanitizable schemes with ad-hoc delegation have been considered in [CLM08, YSL10], and (blockwise) detectability to identify the source of individual message blocks has been introduced in [BPS12]. We consider here the basic case of a single signer and sanitizer only, with the fundamental signature functionality. While the idea of sanitization over encrypted data applies equally well to the aforementioned augmented settings, considering all these cases is beyond the scope of our work.

The notion of signcryption has been introduced by Zheng [Zhe97]. For an introduction to such schemes, we refer to the textbook of Dent and Zheng [DZ10]. Concerning signcryption schemes with advanced operational features, in 2002, Malone-Lee introduced the first *functional* signcryption system in terms of an identity-based signcryption scheme [ML02]. However, apart from this extension, we are unaware of any other functional combinations, especially not of any using some form of malleable signatures.

Recently, other notions of signatures supporting different functionality such as functional signatures [BGI14], policy-based signatures [BF14], delegatable functional signatures [BMS13], and operational signatures [BDF+14] evolved. The latter work provides a general framework for signatures supporting operations which, unlike the former ones, also includes sanitizable signatures. However, they do not work over encrypted data which is the focus of our work here.

2 Preliminaries

In this section, we introduce the basic building blocks for constructing sanitizable signcryption. These primitives comprise (regular) sanitizable signatures as well as multi-input functional encryption and fully homomorphic encryption.

2.1 Notation and Syntax

Throughout this work, we denote the security parameter by λ. Algorithms are usually indicated by a sans-serif font as KeyGen or Enc. We only consider probabilistic polynomial-time (ppt) algorithms referring to those which run in time polynomial in the security parameter and, therefore, often omit the security parameter from the input. We write $m \in \{0,1\}^n$ to describe a string composed of n bits for an integer $n \in \mathbb{N}$. Strings of arbitrary but finite length are denoted by $m \in \{0,1\}^*$. The length of a string m is denoted by $|m|$. If m_1 and m_2 are strings, then $m_1 \| m_2$ describes their concatenation.

We say that a function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is *negligible* if it vanishes faster than any inverse polynomial, i.e., for every polynomial p, there is a bound $N \in \mathbb{N}$ such that for all $\lambda > N$ we have that $\mathsf{negl}(\lambda) < \frac{1}{p(\lambda)}$. We will always use negl to describe a negligible function.

Let E be an event, then $\Pr[E]$ denotes the probability of event E happening. Conditional probabilities are denoted by $\Pr[A \mid B]$ which can be read as the probability of event A occurring, given that event B has occurred. The probability space over which the probabilities are taken is given in subscript. We write $x \leftarrow S$ to denote that an element x is chosen uniformly at random from a set S.

Examples of sets S are the *message space* $\mathcal{M}$, *key space* $\mathcal{K}$, and *ciphertext space* $\mathcal{C}$. Note that each of these sets may depend on the security parameter or public key, however, to gain readability we have decided not to write $\mathcal{M}_\lambda$ or $\mathcal{M}_{pk}$ but remain with $\mathcal{M}$, respectively. Informally, an oracle can be seen as a black-box function which will return a specific output for given inputs. Let $\mathcal{O}$ be an oracle that takes two inputs and has an internal key k. We denote that an adversary $\mathcal{A}$, running solely on the security parameter 1^λ, has access to this oracle by $\mathcal{A}^{\mathcal{O}(\cdot,\cdot,k)}(1^\lambda)$. Oracles carrying the name of a concrete function will always evaluate this function and not be specified separately. By definition, any oracle query execution requires constant time.

2.2 Sanitizable Signatures

Sanitizable signatures are malleable digital signatures which allow a signer to designate a third party, the so-called *sanitizer*, to alter the original message at some later point and still derive a valid signature verifiable under the signer's public key. During the signing process, the signer determines which parts of the message are modifiable by specifying a description $\textsc{adm} \in \{0,1\}^*$ of admissible parts for the message. In the simplest case, messages are sequences of blocks $m = m_1 \| \ldots \| m_\ell$ with $\ell \in \mathbb{N}$, and the description $\textsc{adm}$ consists of the indices of admissible message blocks which the sanitizer can change.

The sanitization is performed by an additional algorithm, inherent to sanitizable signatures, called Sanit. The sanitizer derives a new valid signature, given the sanitizer's secret key sk_{san}. It alters the message according to a modification description $\textsc{mod} \in \{0,1\}^*$ (if permitted by $\textsc{adm}$), which in the blockwise example above may, for example, be seen as a list of block indices and modified block data, i.e., (j, m'_j). In the context of blockwise sanitization, we say that a modification $\textsc{mod}$ is admissible according to $\textsc{adm}$, if each block index of every pair in $\textsc{mod}$ also appears in $\textsc{adm}$. To enforce compatibility of the desired modifications with $\textsc{adm}$ in general, we will sometimes misuse notation by viewing $\textsc{mod}$ as a function applying the modifications to the given message, i.e., $\textsc{mod}(m) \mapsto m'$ for messages m, m' from the respective message space $\mathcal{M}$, and $\textsc{adm}$ as a predicate indicating whether a modification is admissible or not, i.e., $\textsc{adm}(\textsc{mod}) \in \{0,1\}$.[3]

Apart from the common algorithms for key generation for signer $\mathsf{KeyGen}_{\mathsf{sig}}$ and sanitizer $\mathsf{KeyGen}_{\mathsf{san}}$, signing messages Sign, verifying signatures Verify, and the sanitization functionality Sanit, a sanitizable signature scheme comes with several other algorithms to specify accountability for a message-signature pair, identifying either the signer or the sanitizer as the source of the pair. In the original definition given

[3]Note that we assume the sanitzer's algorithm Sanit always performs a sanity check on the given modification, i.e., whether $\textsc{adm}(\textsc{mod}) = 1$ and that this check can be carried out efficiently in the security parameter.

by [ACdT05], a Proof and Judge algorithm perform this task. The idea is that a signer, given his secret key, can create a proof of origin which can be verified by Judge. However, running Proof required knowledge of the signer's secret key and, thus, his cooperation. To allow any party to determine a signature's origin, Brzuska et al. [BPS12] introduced (non-interactive) public accountability. They discovered that public accountability and transparency, the later requiring that it is infeasible to determine if a message was sanitized or not, are mutually exclusive. Since for example in our semi-oblivious sanitizable signcryption case the encrypted signatures of the signer are easy to distinguish from the signature in clear produced by the sanitizer, as long as both parties behave honestly, determining the origin is straightforward which is why transparency cannot be achieved. This, on the other hand, allows us to revert to public accountability and makes the Proof algorithm obsolete which is achievable, as long as the underlying sanitizable signature scheme is publicly accountable, too. We nonetheless keep it here for sake of generality, but let it output $\perp$ in our concrete solutions.

Definition 2.1 *A* sanitizable signature scheme *is a tuple of ppt algorithms* ($\mathsf{KeyGen}_{\mathsf{sig}}$, $\mathsf{KeyGen}_{\mathsf{san}}$, Sign, Sanit, Verify, Proof, Judge) *where*

Key Generation. *The two key generation algorithms* $\mathsf{KeyGen}_{\mathsf{sig}}$ *and* $\mathsf{KeyGen}_{\mathsf{san}}$ *each take a security parameter* 1^λ *as input and output a pair of public key pk and secret key sk, i.e.,* $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(1^\lambda)$ *and* $(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(1^\lambda)$.

Signing. *The* Sign *algorithm takes as input a message* $m \in \mathcal{M}$*, a signer's secret key* sk_{sig}*, a sanitizer's public key* pk_{san}*, as well as a description* ADM *specifying admissible modifications of the message. It outputs a message-signature pair* (m, σ) *where we assume that* ADM *is recoverable from any valid pair given the corresponding sanitizer's secret key* sk_{san} *(or* $\perp$ *in case of an error), i.e.,* $(m, \sigma) \leftarrow \mathsf{Sign}(m, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathrm{ADM})$.

Sanitization. *The* Sanit *algorithm takes as input a message-signature pair* (m, σ)*, a signer's public key* pk_{sig}*, a sanitizer's secret key* sk_{san}*, as well as a description* MOD *of intended modifications. The algorithm first checks whether* σ *is a valid signature for* m*, then, retrieves* ADM *from the given message-signature pair, checks whether* MOD *is admissible, and, if so, modifies the message accordingly to* m'*. Finally, it outputs a derived message-signature pair* (m', σ')*, i.e.,* $(m', \sigma') \leftarrow \mathsf{Sanit}((m, \sigma), pk_{\mathsf{sig}}, sk_{\mathsf{san}}, \mathrm{MOD})$.

Verification. *The* Verify *algorithm takes as input a message-signature pair* (m, σ) *as well as both a signer's and a sanitizer's public key* pk_{sig} *and* pk_{san}*. It outputs a bit* $b \in \{0, 1\}$ *indicating whether the signature is valid (in case* $b = 1$*) or not (if* $b = 0$*). We require* Verify *to be deterministic and, hence, write* $\mathsf{Verify}((m, \sigma), pk_{\mathsf{sig}}, pk_{\mathsf{san}}) = b$.

Proof. *The* Proof *algorithm takes as input a signer's secret key* sk_{sig}*, a message-signature pair* (m, σ)*, a set of polynomially many message-signature pairs* $\{(m_i, \sigma_i) \mid i \in \{1, \ldots, t\}\}$*, as well as a sanitizer's public key* pk_{san}*. It outputs a string* $\pi \in \{0, 1\}^* \cup \{\perp\}$ *called* proof *which might be empty (if* $\pi = \perp$*), i.e.,* $\pi \leftarrow \mathsf{Proof}(sk_{\mathsf{sig}}, (m, \sigma), \{(m_i, \sigma_i) \mid i \in \{1, \ldots, t\}\}, pk_{\mathsf{san}})$.

Judging. *The* Judge *algorithm takes as input valid a message-signature pair* (m, σ)*, a signer's and sanitizer's public key* pk_{sig} *and* pk_{san}*, as well as a proof* $\pi \in \{0, 1\}^* \cup \{\perp\}$*. It outputs a decision* $b \in \{0, 1\}$ *indicating whether the signer (if* $b = 1$*) or sanitizer (if* $b = 0$*) generated the given message-signature pair (or* $\perp$ *in case of an error). We require* Judge *to be deterministic and, hence, write* $\mathsf{Judge}((m, \sigma), pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \pi) = b$.

We require the scheme to achieve the following correctness properties:

Perfect signer correctness. *For all security parameters* $\lambda \in \mathbb{N}$*, all signer's key pairs* $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(1^\lambda)$*, all sanitizer's key pairs* $(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(1^\lambda)$*, all messages* $m \in \mathcal{M}$*, all descriptions* $\text{ADM} \in \{0,1\}^*$*, and all signatures* $\sigma \leftarrow \mathsf{Sign}(m, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \text{ADM})$ *we have*

$$\mathsf{Verify}((m,\sigma), pk_{\mathsf{sig}}, pk_{\mathsf{san}}) = 1.$$

Perfect sanitizer correctness. *For all security parameters* $\lambda \in \mathbb{N}$*, all signer's key pairs* $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(1^\lambda)$*, all sanitizer's key pairs* $(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(1^\lambda)$*, all messages* $m \in \mathcal{M}$*, all signatures* σ *such that* $\mathsf{Verify}((m,\sigma), pk_{\mathsf{sig}}, pk_{\mathsf{san}}) = 1$*, all descriptions* $\text{MOD} \in \{0,1\}^*$ *such that* MOD *matches the* ADM *recoverable from* (m,σ)*, i.e.,* $\text{ADM}(\text{MOD}) = 1$*, and all derived message-signature pairs* $(m',\sigma') \leftarrow \mathsf{Sanit}((m,\sigma), pk_{\mathsf{sig}}, sk_{\mathsf{san}}, \text{MOD})$ *such that* $\text{MOD}(m) = m'$ *we have*

$$\mathsf{Verify}((m',\sigma'), pk_{\mathsf{sig}}, pk_{\mathsf{san}}) = 1.$$

Perfect proof correctness. *For all security parameters* $\lambda \in \mathbb{N}$*, all signer's key pairs* $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(1^\lambda)$*, all sanitizer's key pairs* $(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(1^\lambda)$*, all messages* $m \in \mathcal{M}$*, all descriptions* $\text{ADM} \in \{0,1\}^*$*, all signatures* $\sigma \leftarrow \mathsf{Sign}(m, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \text{ADM})$*, all descriptions* $\text{MOD} \in \{0,1\}^*$ *which match* ADM*, i.e., with* $\text{ADM}(\text{MOD}) = 1$*, all derived signatures* $(m',\sigma') \leftarrow \mathsf{Sanit}((m,\sigma), pk_{\mathsf{sig}}, sk_{\mathsf{san}}, \text{MOD})$ *such that* $\mathsf{Verify}((m',\sigma'), pk_{\mathsf{sig}}, pk_{\mathsf{san}}) = 1$ *and* $\text{MOD}(m) = m'$*, any (polynomially many) messages* $m_1, \ldots, m_t \in \mathcal{M}$ *with descriptions* $\text{ADM}_1, \ldots, \text{ADM}_t \in \{0,1\}^*$ *and signatures* $\sigma_i \leftarrow \mathsf{Sign}(m_i, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \text{ADM}_i)$ *such that* $(m,\sigma) = (m_i,\sigma_i)$ *holds for some index* $i \in \{1,\ldots,t\}$*, and all proofs* $\pi \leftarrow \mathsf{Proof}(sk_{\mathsf{sig}}, (m',\sigma'), \{(m_i,\sigma_i) \mid i \in \{1,\ldots,t\}\}, pk_{\mathsf{san}})$ *we have*

$$\Pr\left[\mathsf{Judge}((m',\sigma'), pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \pi) = 1\right] = 0\ .$$

We say that the scheme is length-invariant *if the size of (sanitized and original) signatures only depends on the security parameter.*

Ateniese et al. [ACdT05] described several desirable security properties which where later formalized by Brzuska et al. [BFF+09, BFLS10]. The basic set of properties consists of:

Unforgeability: It should be infeasible to forge a new message-signature pair for outsiders.

Immutability: It should be infeasible for a sanitizer to modify inadmissible parts of the message.

Privcacy: It should be infeasible to learn the original content of modified message parts.

Unlinkability: It should be infeasible to link sanitized messages originating from the same source.

Transparency: It should be infeasible to determine whether a message was sanitized or not.

Accountability: It should be infeasible to blame the opposite party for a signature one has created.

Some of them come in strong and weak versions, depending on the oracles available to the adversary. Since we give the formal definitions for sanitizable signcryption schemes in Appendix A, and the ones for regular sanitizable signatures are easy to derive, we omit definitions of the above properties here.

2.3 Multi-input Functional Encryption

In [GGG+14], Goldwasser et al. introduced the notion of *multi-input functional encryption*, which extends the original functional encryption scheme to support multi-variate functions, i.e., supporting functions which take more than one input.

Definition 2.2 *Let $\mathcal{F}$ be a family of n-ary functions. A* multi-input functional encryption scheme *for $\mathcal{F}$ is a tuple of four ppt algorithms (*$\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec}$*) where*

> ***Setup.*** *The* Setup *algorithm takes an integer $n \in \mathbb{N}$, representing the function's arity, as input and outputs a master secret key* Msk *as well as a sequence of n encryption keys $\mathbf{EK} = (\mathsf{EK}_1, \ldots, \mathsf{EK}_n)$, i.e., $(\mathsf{Msk}, \mathbf{EK}) \leftarrow \mathsf{Setup}(n)$.*
>
> ***Encryption.*** *The* Enc *algorithm takes as input an encryption key $\mathsf{EK} \in \mathbf{EK}$ and a message $m \in \mathcal{M}$. It outputs a ciphertext $c \in \mathcal{C}$, i.e., $c \leftarrow \mathsf{Enc}(\mathsf{EK}, m)$.*
>
> ***Key Generation.*** *The* KeyGen *algorithm takes the master secret key* Msk *and an n-ary function $f \in \mathcal{F}$ as input. It outputs a secret key sk_f for the function f, i.e., $sk_f \leftarrow \mathsf{KeyGen}(\mathsf{Msk}, f)$.*
>
> ***Decryption.*** *The* Dec *algorithm takes as input a secret key sk_f and n ciphertexts $c_1, \ldots, c_n$ such that c_i was encrypted using the i-th encryption key EK_i for every $i \in \{1, \ldots, n\}$. Finally, it outputs a string z (or $\perp$ in case of an error). We require* Dec *to be deterministic, i.e., $\mathsf{Dec}(sk_f, c_1, \ldots, c_n) = z$.*

We require the scheme to have perfect correctness*, i.e., for all $n \in \mathbb{N}$, all $f \in \mathcal{F}$, all $(\mathsf{Msk}, (\mathsf{EK}_1, \ldots, \mathsf{EK}_n)) \leftarrow \mathsf{Setup}(n)$, all $m_1, \ldots, m_n \in \mathcal{M}$, all $c_1 \leftarrow \mathsf{Enc}(\mathsf{EK}_1, m_1), \ldots, c_n \leftarrow \mathsf{Enc}(\mathsf{EK}_n, m_n)$, and all $sk_f \leftarrow \mathsf{KeyGen}(\mathsf{Msk}, f)$ we have*

$$\mathsf{Dec}(sk_f, c_1, \ldots, c_n) = f(m_1, \ldots, m_n) \ .$$

Note that, unlike [GGG+14], we have decided to assume *perfect* correctness, whereas Goldwasser et al. allow for a negligible error. The reason is to simplify later proofs. However, all proofs presented equally hold given schemes with negligible error probability with slight but negligible loss in the success probability of the adversary.

Naturally, the security notions for general public-key encryption schemes can be adopted to fit the advanced primitive of functional encryption. Goldwasser et al. formalize both indistinguishability-based and simulation-based security notions for multi-input functional encryption schemes. Since the notions we will be using are solely of the indistinguishability-based flavor, we limit ourselves to these notions and refer to [GGG+14] for a detailed analysis on further security notions as well as their relations and feasibility.

The definition of indistinguishability-based security for multi-input functional encryption Goldwasser et al. give only allows for non-adaptive queries to the challenger, i.e., the attacker needs to specify all challenge queries before seeing any of the encrypted values. A more general definition, which we needed for our proofs later, is to allow for adaptive queries. Therefore, we state an adaptive version in the following. Note that it can be shown by a simple hybrid argument, that the two notions are equivalent.

Goldwasser et al. show that the notion of full indistinguishability-based security —given all encryption keys as well as polynomial challenges and decryption keys— is achievable under the assumption of sub-exponential secure indistinguishability obfuscators for general circuits, for which a candidate construction was given in [GGH+13], and sub-exponentially secure one-way functions. This matches our requirements. For more details on the achievablity of this notion, we refer to [GGG+14].

Definition 2.3 *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *be a multi-input functional encryption scheme for the n-ary function family $\mathcal{F}$. We say that* FE *is* miIND-secure *if for all ppt adversaries $\mathcal{A}$ there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{miIND}}_{\mathcal{A},\mathsf{FE}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of $\mathcal{A}$ and the miIND *experiment which is defined as follows:*

$\mathsf{Exp}^{\mathsf{milND}}_{\mathcal{A},\mathsf{FE}}(\lambda)$

$I \leftarrow \mathcal{A}(1^\lambda)$;
$(\mathsf{Msk}, \mathbf{EK}) \leftarrow \mathsf{Setup}(n)$;
$b \leftarrow \{0,1\}$;
$b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{Msk},\cdot),\mathsf{LoREnc}(\mathbf{EK},\cdot,\cdot,b)}(\mathbf{EK}_I)$;
if $(b = b' \wedge$ *"compatibilities okay"*$)$ **then**
  return 1.
else return 0.

$\mathsf{LoREnc}(\mathbf{EK}, m_0, m_1, b)$

for $i = 1, \ldots, n$ **do**
  if $|m_{0,i}| \neq |m_{1,i}|$ **then**
    return $\perp$.
  else
    $\mathsf{EK}_i \leftarrow \mathbf{EK}$;
    $c_{b,i} \leftarrow \mathsf{Enc}(\mathsf{EK}_i, m_{b,i})$;
return $(c_{b,1}, \ldots, c_{b,n})$.

where $\mathbf{EK}_I = (\mathsf{EK}_i)_{i \in I}$ *and* LoREnc *takes two messages of the form* $m_0 = (m_{0,1}, \ldots, m_{0,n})$ *and* $m_1 = (m_{1,1}, \ldots, m_{1,n})$. *The requirement "compatibilities okay" demands the following:*

> ***Function compatibility****, requiring that any two message vectors* m_0 *and* m_1 *queried to the* LoREnc *oracle cannot be split by any function* f *for which* $\mathcal{A}$ *has queried its* KeyGen *oracle. Here, splitting means that for any possible input* $(x_1, \ldots, x_n)$ *for* f*, it does not matter whether elements of* $m_{0,i}$ *or* $m_{1,i}$ *are used as* x_i *in any subset of the positions specified by the set* I*; the function values still concur on all these inputs. As consequence, only challenge queries can be asked which cannot be split by any of the currently held decryption keys.*
>
> ***Message compatibility****, requiring that any function* f *queried to the* KeyGen *oracle cannot split any of the challenge message vectors queried.*

For a formal definition of function and message compatibility, we refer to [GGG+14].

Even though the above definition is stated in great generality, we will only consider the case when *all* encryption keys are handed to the adversary. This is also referred to as the "public-key setting". Lastly, we want to remark that there is a second important security property for functional encryption schemes, namely *function privacy* [SSW09, BRS13, AAB+13, BS15]. Intuitively, this notion requires that a decryption key sk_f does not leak any information about the function f. Since we will not need to keep the decryption key functions secret for our construction, we will not elaborate further on this property.

2.4 Fully Homomorphic Encryption

A fully homomorphic encryption scheme can be seen as a public-key encryption scheme which permits any kind of efficient (arithmetical) operation on encrypted data by offering an additional evaluation algorithm which, unlike functional encryption, yields encrypted results after evaluation.

Definition 2.4 *A* fully homomorphic encryption scheme *is a tuple of four ppt algorithms* (KeyGen, Enc, Dec, Eval) *where*

> ***Key Generation.*** *The* KeyGen *algorithm takes a security parameter* 1^λ *as input and outputs a pair of public key* pk *and secret key* sk*, i.e.,* $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
>
> ***Encryption.*** *The* Enc *algorithm takes as input a public key* pk *and a message* $m \in \mathcal{M}$*. It outputs a ciphertext* $c \in \mathcal{C}$*, i.e.,* $c \leftarrow \mathsf{Enc}(pk, m)$.
>
> ***Decryption.*** *The* Dec *algorithm takes as input a secret key* sk *and a ciphertext* $c \in \mathcal{C}$*. It outputs a message* $m \in \mathcal{M}$ *(or* $\perp$ *in case of an error). We require* Dec *to be deterministic and, hence, write* $\mathsf{Dec}(sk, c) = m$.
>
> ***Evaluate.*** *The* Eval *algorithm takes as input a public key* pk*, a circuit* C_n *for* n *inputs, and* n *ciphertexts* $c_1, \ldots, c_n$*. It outputs a ciphertext* $c' \leftarrow \mathsf{Eval}(pk, C_n, c_1, \ldots, c_n)$.

We require the scheme to achieve perfect correctness*, i.e., for all* $n \in \mathbb{N}$*, all circuits* C_n*, all* $\lambda \in \mathbb{N}$*, all* $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, all* $m_1, \ldots, m_n \in \mathcal{M}$*, all* $c_1 \leftarrow \mathsf{Enc}(pk, m_1), \ldots, c_n \leftarrow \mathsf{Enc}(pk, m_n)$*, and all* $c' \leftarrow \mathsf{Eval}(pk, C_n, c_1, \ldots, c_n)$ *we have*

$$\mathsf{Dec}(sk, c') = C_n(m_1, \ldots, m_n).$$

To exclude the trivial scheme where the Eval algorithm just strings the description of the circuit together with the ciphertext and leaves the decryption algorithm to manage the entire evaluation itself, we require fully homomorphic encryption schemes to be *compactly evaluable* and *circuit private* [Gen09]. The security notions for fully homomorphic encryption schemes can be similarly defined as for regular encryption schemes, and likewise, we refer to Gentry for a formal treatment thereof.

3 Semi-Oblivious Sanitizable Signcryption

As mentioned in the introduction, there are two scenarios for sanitizable signcryption, depending on whether the sanitizer may see the sanitized message or not. In this section, we address the first case, namely when sanitization and decryption are interwoven.

3.1 The Primitive

Here, we define the new primitive of *semi-oblivious sanitizable signcryption*. The scenario we have in mind is the one depicted in Figure 1 and consists of two stages. In the first stage, with data flow from signer to sanitizer, a message is signed and the pair is encrypted to ensure integrity, authenticity, as well as confidentiality at this point. In the second stage, the sanitizer modifies the encrypted message-signature pair "through the encryption" and, simultaneously, decrypts the resulting data to get a pair in clear. This derived pair can now be forwarded to any designated receivers.

As for general signcryption schemes, we are equipped with a setup algorithm and two key generation algorithms for sender (in our case, the signer) and receiver (here, the sanitizer). The SignEnc algorithm takes care of the first phase of our scenario by combining the generation of a signature with an encryption of the tuple. Only after this has been performed, the SanDec algorithm will be able to handle the second phase and both sanitize and then decrypt the message-signature pair. Note that this initially limits the sanitization process to encrypted tuple only. Nevertheless, in our generic construction, using the publicly available encryption keys of the functional encryption scheme, the sanitizer can re-encrypt a sanitized message-signature pair and then repeat the sanitization process to achieve multiple sanitization, if required, but it is not a property native to the primitive itself.

Next, we discuss the additional algorithms —Verify, Proof, and Judge— which stem from the accountability requirement of sanitizable schemes and augment traditional signcryption schemes. The main difference to the case of regular sanitizable schemes is that these algorithms need to be able to process encrypted data, too. We usually write $\widetilde{\mathsf{m}\sigma}$ indicating either an encrypted or unencrypted message-signature pair (m, σ) (and, accordingly, $\widetilde{\mathsf{m}^*\sigma^*}$ if we have an associated pair (m^*, σ^*)). Note that, for our construction, we achieve publicly evaluable versions for Verify and Judge, and since our Proof algorithm always outputs $\perp$, we also obtain a trivial publicly evaluable algorithm Proof, too.

Note that working over encrypted data causes some interesting practical challenge. Usually, the description ADM of the set of admissible operations on a message-signature pair can be read off the signature, given the sanitizer's secret key. Here, the signature is now encrypted as it may otherwise leak information about the message. This does not directly cause any problem since the adversary usually decides which modification MOD should be performed, and the validity is checked still implicitly by the sanitation algorithm. In practice, usability might be more difficult since the sanitizer needs to be told which modifications

should be performed. If ADM should be available to the sanitizer, one can attach ADM in clear to the ciphertext, but should be aware that this information is not authenticated then (unless one adds another regular signature on top).

Definition 3.1 *A* semi-oblivious sanitizable signcryption scheme *is a tuple of ppt algorithms* $\mathsf{sOSSC} = (\mathsf{Setup}, \mathsf{KeyGen_{sig}}, \mathsf{KeyGen_{san}}, \mathsf{SignEnc}, \mathsf{SanDec}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *such that*

Setup. *Upon input of the security parameter* 1^λ*, the algorithm outputs a master secret key* Msk *as well as public parameters* pparam*, i.e.,* $(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$.

Key Generation (Signer). *The algorithm* $\mathsf{KeyGen_{sig}}$ *takes as input the master secret key* Msk *and outputs a signer's public and secret key, i.e.,* $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen_{sig}}(\mathsf{Msk})$.

Key Generation (Sanitizer). *The algorithm* $\mathsf{KeyGen_{san}}$ *takes as input the master secret key* Msk *and outputs a sanitizer's public and secret key, i.e.,* $(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen_{san}}(\mathsf{Msk})$.

SignEnc. *Upon input of a message* m*, a description* ADM *of the admissible parts, a signer's secret key* sk_{sig}*, a sanitizer's public key* pk_{san}*, as well as public parameters* pparam*, the algorithm* SignEnc *outputs an encrypted message-signature pair (or* $\perp$ *in case of an error), i.e.,* $c_{(m,\sigma)} \leftarrow \mathsf{SignEnc}(m, \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam})$.

SanDec. *Upon input of an encrypted message-signature pair* $c_{(m,\sigma)}$*, a description of desired modifications* MOD*, a sanitizer's secret key* sk_{san}*, a signer's public key* pk_{sig}*, and public parameters* pparam*, the algorithm* SanDec *outputs a decrypted and sanitized version of a message-signature pair (or* $\perp$ *in case of an error), i.e.,* $(m', \sigma') \leftarrow \mathsf{SanDec}(c_{(m,\sigma)}, \mathrm{MOD}, sk_{\mathsf{san}}, pk_{\mathsf{sig}}, \mathsf{pparam})$.

Verify. *Upon input of a message-signature pair* $\widetilde{\mathsf{m}\sigma}$*, either in clear* (m, σ) *or encrypted* $c_{(m,\sigma)}$*, public keys of both signer* pk_{sig} *and sanitizer* pk_{san}*, as well as public parameters* pparam*, the algorithm* Verify *returns a bit* b *indicating whether the (encrypted) message-signature pair is valid or not. We require* Verify *to be deterministic and, hence, write* $\mathsf{Verify}(\widetilde{\mathsf{m}\sigma}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam}) = b$.

Proof. *Upon input of a signer's secret key* sk_{sig}*, a possibly encrypted message-signature pair* $\widetilde{\mathsf{m}\sigma}$*, a possibly empty list of (possibly encrypted) message-signature pairs* $\{\widetilde{\mathsf{m}_1\sigma_1}, \ldots, \widetilde{\mathsf{m}_\mathsf{t}\sigma_\mathsf{t}}\}$*, a sanitizer's public key* pk_{san}*, and public parameters* pparam*, the algorithm* Proof *outputs a string* $\pi \in \{0,1\}^* \cup \{\perp\}$*, i.e.,* $\pi \leftarrow \mathsf{Proof}(sk_{\mathsf{sig}}, \widetilde{\mathsf{m}\sigma}, \{\widetilde{\mathsf{m}_1\sigma_1}, \ldots, \widetilde{\mathsf{m}_\mathsf{t}\sigma_\mathsf{t}}\}, pk_{\mathsf{san}}, \mathsf{pparam})$.

Judge. *Upon input of a possibly encrypted message-signature pair* $\widetilde{\mathsf{m}\sigma}$*, the public keys of both signer* pk_{sig} *and sanitizer* pk_{san}*, a string* $\pi \in \{0,1\}^* \cup \{\perp\}$*, as well as public parameters* pparam*, the algorithm* Judge *outputs a bit* d *indicating which party has generated the given pair (or* $\perp$ *in case of an error). Here,* $d = 1$ *refers to the signer and* $d = 0$ *to the sanitizer. We require* Judge *to be deterministic and, hence, write* $\mathsf{Judge}(\widetilde{\mathsf{m}\sigma}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \pi, \mathsf{pparam}) = d$.

We expect a semi-oblivious sanitizable signcryption scheme to have the following correctness properties:

Perfect signer correctness. *For all security parameters* $\lambda \in \mathbb{N}$*, all system instantiations* $(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$*, all signer's key pairs* $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen_{sig}}(\mathsf{Msk})$*, all sanitizer's key pairs* $(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen_{san}}(\mathsf{Msk})$*, all messages* $m \in \mathcal{M}$*, all descriptions* $\mathrm{ADM} \in \{0,1\}^*$*, and all encrypted signatures* $(c_{(m,\sigma)}, \mathrm{ADM}) \leftarrow \mathsf{SignEnc}(m, \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam})$ *we have*

$$\mathsf{Verify}(c_{(m,\sigma)}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam}) = 1 \ .$$

Perfect sanitizer correctness. *For all security parameters* $\lambda \in \mathbb{N}$*, all system instantiations* $(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$*, all signer's key pairs* $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(\mathsf{Msk})$*, all sanitizer's key pairs* $(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(\mathsf{Msk})$*, all messages* $m \in \mathcal{M}$*, all descriptions* $\mathrm{ADM} \in \{0,1\}^*$*, all encrypted signatures* $(c_{(m,\sigma)}, \mathrm{ADM}) \leftarrow \mathsf{SignEnc}(m, \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam})$*, all descriptions* $\mathrm{MOD} \in \{0,1\}^*$ *which match* ADM*, and all derived message-signature pairs* $(m', \sigma') \leftarrow \mathsf{SanDec}(c_{(m,\sigma)}, \mathrm{MOD}, sk_{\mathsf{san}}, pk_{\mathsf{sig}}, \mathsf{pparam})$ *with* $\mathrm{MOD}(m) = m'$ *we have*

$$\mathsf{Verify}((m', \sigma'), pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam}) = 1 \ .$$

Perfect proof correctness. *For all security parameters* $\lambda \in \mathbb{N}$*, all system instantiations* $(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$*, all signer's key pairs* $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(\mathsf{Msk})$*, all sanitizer's key pairs* $(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(\mathsf{Msk})$*, all messages* $m \in \mathcal{M}$*, all descriptions* $\mathrm{ADM} \in \{0,1\}^*$*, all encrypted signatures* $(c_{(m,\sigma)}, \mathrm{ADM}) \leftarrow \mathsf{SignEnc}(m, \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam})$*, all descriptions* $\mathrm{MOD} \in \{0,1\}^*$ *which match* ADM*, all derived signatures* $(m', \sigma') \leftarrow \mathsf{SanDec}(c_{(m,\sigma)}, \mathrm{MOD}, sk_{\mathsf{san}}, pk_{\mathsf{sig}}, \mathsf{pparam})$ *such that* $\mathsf{Verify}((m', \sigma'), pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam}) = 1$ *and* $\mathrm{MOD}(m) = m'$*, any (polynomially many) valid, possibly encrypted, message-signature pairs* $\widetilde{\mathsf{m}_1\sigma_1}, \ldots, \widetilde{\mathsf{m}_\mathsf{t}\sigma_\mathsf{t}}$ *such that* $c_{(m,\sigma)} = \widetilde{\mathsf{m}_\mathsf{i}\sigma_\mathsf{i}}$ *holds for some* $i \in \{1, \ldots, t\}$*, and all proofs* $\pi \leftarrow \mathsf{Proof}(sk_{\mathsf{sig}}, (m', \sigma'), \{\widetilde{\mathsf{m}_1\sigma_1}, \ldots, \widetilde{\mathsf{m}_\mathsf{t}\sigma_\mathsf{t}}\}, pk_{\mathsf{san}}, \mathsf{pparam})$ *we have*

$$\mathsf{Judge}((m', \sigma'), pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \pi, \mathsf{pparam}) = 0 \ .$$

3.2 The Construction

The construction we present in the following is a generic composition of a sanitizable signature scheme with a multi-input functional encryption scheme. Roughly, the idea is to use the functional encryption scheme to perform operations on the ciphertexts. More concretely, the signer signs a message m (together with the description ADM of admissible operations) to get a signature σ, and then encrypts the data under the functional encryption scheme. Here, we assume as in Definition 2.1 that the underyling sanitizable scheme is length-invariant, i.e., all signatures, including the descriptions of ADM, are of equal length, depending only on the security parameter, such that the encryption of the signatures does not leak information about the message through the ciphertext length. The sanitizer is given a secret decryption key $\mathsf{sk}_{\mathsf{Sanit}}$ corresponding to the function Sanit which applies the modifications MOD to get the message m' and derive a new signature σ' under the underlying sanitizable scheme.

Note that we construct decryption keys for *probabilistic* functions. However, current (multi-input) functional encryption constructions, usually based upon *indistinguishability obfuscation*, only support deterministic functions. Nevertheless, we can "derandomize" the functions using the inputs to generate randomness, which is done be passing these through a pseudorandom function and utilizing the pseudorandom output as random coins for the probabilistic function in question. See [CLTV14] for further details. An alternative is to use sanitizable signature schemes which have deterministic algorithms, more precisely such that Verify, Judge, and Sanit are deterministic. This might, however, impact the choice of candidates or which security property are achievable for this primitive.

Construction 3.2 *Assume a sanitizable signature scheme* $\mathcal{S} = (\mathsf{S.KeyGen}_{\mathsf{sig}}, \mathsf{S.KeyGen}_{\mathsf{san}}, \mathsf{S.Sign}, \mathsf{S.Sanit}, \mathsf{S.Verify}, \mathsf{S.Proof}, \mathsf{S.Judge})$ *as well as a multi-input functional encryption scheme* $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.KeyGen}, \mathsf{FE.Dec})$*. Then we define* $\mathsf{sOSSC} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{sig}}, \mathsf{KeyGen}_{\mathsf{san}}, \mathsf{SignEnc}, \mathsf{SanDec}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *as follows.*

> ***Setup.*** *Upon input of the security parameter* 1^λ*, the algorithm initializes the functional encryption scheme and computes decryption keys* $\mathsf{sk}_{\mathsf{Vf}}$ *and* sk_{J} *for the publicly evaluable functions* $\mathsf{S.Verify}$ *and* $\mathsf{S.Judge}$*, respectively. Finally, it outputs a master secret key* Msk *as well as public parameters* pparam*, consisting of the encryption keys* $\mathbf{EK} = (\mathsf{EK}_1, \ldots, \mathsf{EK}_4)$ *as well as the computed decryption keys.*

$\underline{\mathsf{Setup}(1^\lambda):}$

$(\mathsf{Msk}, \mathbf{EK}) \leftarrow \mathsf{FE.Setup}(4);$
$\mathsf{sk}_{\mathsf{Vf}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{Msk}, \mathsf{S.Verify}(\cdot,\cdot,\cdot));$
$\mathsf{sk}_{\mathsf{J}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{Msk}, \mathsf{S.Judge}(\cdot,\cdot,\cdot,\cdot));$
$\mathsf{pparam} \leftarrow (\mathbf{EK}, \mathsf{sk}_{\mathsf{Vf}}, \mathsf{sk}_{\mathsf{J}});$
return $(\mathsf{Msk}, \mathsf{pparam})$.

KeyGen (Signer). *The algorithm* $\mathsf{KeyGen}_{\mathsf{sig}}$ *takes as input the master secret key* Msk*, generates a signer's key pair for* $\mathcal{S}$*, as well as a universal decryption key for the identity function* $\mathsf{ID}(\cdot)$ *and outputs these.*

$\underline{\mathsf{KeyGen}_{\mathsf{sig}}(\mathsf{Msk}):}$

$(sk'_{\mathsf{sig}}, pk_{\mathsf{sig}}) \leftarrow \mathsf{S.KeyGen}_{\mathsf{sig}}(1^\lambda);$
$\mathsf{sk}_{\mathsf{ID}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{Msk}, \mathsf{ID}(\cdot));$
$sk_{\mathsf{sig}} \leftarrow (sk'_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{ID}});$
return $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}})$.

KeyGen (Sanitizer). *The algorithm* $\mathsf{KeyGen}_{\mathsf{san}}$ *takes as input the master secret key* Msk*, generates a sanitizer's key pair for* $\mathcal{S}$*, as well as decryption key* $\mathsf{sk}_{\mathsf{Sanit}}$ *for the function* S.Sanit*, with the sanitizer's secret key hard-coded inside, and outputs these.*

$\underline{\mathsf{KeyGen}_{\mathsf{san}}(\mathsf{Msk}):}$

$(sk'_{\mathsf{san}}, pk_{\mathsf{san}}) \leftarrow \mathsf{S.KeyGen}_{\mathsf{san}}(1^\lambda);$
$\mathsf{sk}_{\mathsf{Sanit}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{Msk}, \mathsf{S.Sanit}(\cdot,\cdot, sk'_{\mathsf{san}}, \cdot));$
$sk_{\mathsf{san}} \leftarrow \mathsf{sk}_{\mathsf{Sanit}};$
return $(pk_{\mathsf{san}}, sk_{\mathsf{san}})$.

SignEnc. *Upon input of a message* m*, a description* ADM *of admissible parts, a signer's secret key* sk_{sig}*, a sanitizer's public key* pk_{san}*, as well as public parameters* pparam*, the algorithm* SignEnc *uses the sanitizable signature scheme to compute a signature* σ *on* m*, then encrypts both message and signature to* $c_{(m,\sigma)}$ *using* FE *under* EK_1 *and returns this value (or* $\perp$ *in case of an error).*

$\underline{\mathsf{SignEnc}(m, \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam}):}$

$sk'_{\mathsf{sig}} \leftarrow sk_{\mathsf{sig}};$
$(m, \sigma) \leftarrow \mathsf{S.Sign}(m, sk'_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathrm{ADM});$
$\mathsf{EK}_1 \leftarrow \mathsf{pparam};$
$c_{(m,\sigma)} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_1, (m, \sigma));$
return $c_{(m,\sigma)}$.

SanDec. *Upon input of an encrypted message-signature pair* $c_{(m,\sigma)}$*, a description of desired modifications* MOD*, a sanitizer's secret key* sk_{san}*, a signer's public key* pk_{sig}*, and public parameters* pparam*, the algorithm* SanDec *uses* FE.Dec *with* $\mathsf{sk}_{\mathsf{Sanit}}$ *on the tuple as well as encryptions under* EK_2 *and* EK_3 *of the remaining inputs* pk_{sig} *and* MOD*. It finally returns the sanitized pair* (m', σ') *output by the decryption procedure (or* $\perp$ *in case of an error).*

$\underline{\mathsf{SanDec}(c_{(m,\sigma)}, \mathrm{MOD}, sk_{\mathsf{san}}, pk_{\mathsf{sig}}, \mathsf{pparam}) :}$

$\mathsf{sk}_{\mathsf{Sanit}} \leftarrow sk_{\mathsf{san}}$;

$(\mathsf{EK}_2, \mathsf{EK}_3) \leftarrow \mathsf{pparam}$;

$c_{pk} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_2, pk_{\mathsf{sig}})$;

$c_{\mathrm{MOD}} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_3, \mathrm{MOD})$;

$(m', \sigma') \leftarrow \mathsf{FE.Dec}(\mathsf{sk}_{\mathsf{Sanit}}, c_{(m,\sigma)}, c_{pk}, c_{\mathrm{MOD}})$;

return (m', σ').

Verify. *Upon input of a (possibly encrypted) message-signature pair* $\widetilde{\mathsf{m}\sigma}$*, public keys of both signer* pk_{sig} *and sanitizer* pk_{san}*, as well as public parameters* pparam*, for an unencrypted message-signature pair the algorithm* Verify *calls upon* S.Verify *to output a bit* b *indicating if the pair is valid or not.*[4] *For an encrypted pair, it calls* FE.Dec *with* $\mathsf{sk}_{\mathsf{Vf}}$ *to output its decision (or* $\perp$ *in case of another error).*

$\underline{\mathsf{Verify}(\widetilde{\mathsf{m}\sigma}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam}) :}$

if $\mathsf{S.Verify}(\widetilde{\mathsf{m}\sigma}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}) == 1$ **then**

  return 1.

else

  $(\mathsf{sk}_{\mathsf{Vf}}, \mathsf{EK}_2, \mathsf{EK}_3) \leftarrow \mathsf{pparam}$;

  $c_{pk_{\mathsf{sig}}} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_2, pk_{\mathsf{sig}})$;

  $c_{pk_{\mathsf{san}}} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_3, pk_{\mathsf{san}})$;

  if $\mathsf{FE.Dec}(\mathsf{sk}_{\mathsf{Vf}}, \widetilde{\mathsf{m}\sigma}, c_{pk_{\mathsf{sig}}}, c_{pk_{\mathsf{san}}}) == 1$ **then**

    return 1.

  else return 0.

Proof. *The* Proof *algorithm always returns an empty proof* $\perp$*.*

$\underline{\mathsf{Proof}(sk_{\mathsf{sig}}, \widetilde{\mathsf{m}\sigma}, \{\widetilde{\mathsf{m}_1\sigma_1}, \ldots, \widetilde{\mathsf{m}_t\sigma_t}\}, pk_{\mathsf{san}}, \mathsf{pparam}) :}$

return $\perp$.

Judge. *Upon input of a (possibly encrypted) message-signature pair* $\widetilde{\mathsf{m}\sigma}$*, public keys of both signer* pk_{sig} *and sanitizer* pk_{san}*, a string* $\pi \in \{0,1\}^* \cup \{\perp\}$*, as well as public parameters* pparam*, the algorithm* Judge *calls upon* S.Judge*, resp.* FE.Dec *with* sk_{J} *in case the pair is encrypted, to output a bit* d *indicating which party has generated the given pair (or* $\perp$ *in case of an error).*

[4] For simplicity and correctness, we assume that message space and ciphertext space are disjoint and S.Verify will always return $\perp$ upon an encrypted tuple. This is easily achievable, for example, by prepending 1 to any ciphertext and 0 for each plaintext.

$\mathsf{Judge}(\widetilde{\mathsf{m}\sigma}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \pi, \mathsf{pparam})$:

if $\mathsf{S.Judge}(\widetilde{\mathsf{m}\sigma}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \pi) == 0$ **then**
  return 0.
else if $\mathsf{S.Judge}(\widetilde{\mathsf{m}\sigma}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \pi) == 1$ **then**
  return 1.
else
  $(\mathsf{sk_J}, \mathsf{EK}_2, \mathsf{EK}_3, \mathsf{EK}_4) \leftarrow \mathsf{pparam}$;
  $c_{pk_{\mathsf{sig}}} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_2, pk_{\mathsf{sig}})$;
  $c_{pk_{\mathsf{san}}} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_3, pk_{\mathsf{san}})$;
  $c_\pi \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_4, \pi)$;
  if $\mathsf{FE.Dec}(\mathsf{sk_J}, \widetilde{\mathsf{m}\sigma}, c_{pk_{\mathsf{sig}}}, c_{pk_{\mathsf{san}}}, c_\pi) == 1$ **then**
    return 1.
  else if $\mathsf{FE.Dec}(\mathsf{sk_J}, \widetilde{\mathsf{m}\sigma}, c_{pk_{\mathsf{sig}}}, c_{pk_{\mathsf{san}}}, c_\pi) == 0$ **then**
    return 0.
  else return $\perp$.

It can be easily formally verified that sOSSC is indeed a semi-oblivious sanitizable signcryption scheme.

A couple of remarks are in place. Firstly, the definition of $\mathsf{KeyGen_{sig}}$ may at first seem unusual since a universal decryption key sk_{ID} is part of the output. The reason is that we want the signing party to be able to decrypt any signcrypted messages at a later point in time. For example, if the scheme is being used for storing files on a server, the file owners might want to perform modifications or review these files, especially the original versions before sanitization.

A second point is that we hard-coded sk'_{san} directly into the Sanit circuit when generating the decryption key $\mathsf{sk_{Sanit}}$. This yields a unique decryption key per sanitizer. It would have been possible to generate a more general decryption key $\mathsf{sk_{Sanit}}$ in the Setup procedure and to slim the $\mathsf{KeyGen_{san}}$ algorithm down to the one of the underlying sanitizable signature scheme. Note that this does not necessarily hide sk_{san} from the sanitizer (which is not necessary, but would be possible if the functional encryption scheme was function private). Hard-coding the key into the circuit limits adversaries in testing different keys for the Sanit circuit.

Recall that as mentioned earlier we wish to achieve non-interactive accountability [BPS12] and therefore manage with a Proof algorithm which solely returns $\perp$.

Finally, a small remark about the amount of distinct decryption keys used in the scheme. In a multi-input functional encryption scheme, we have distinct encryption keys for each position of the input, i.e., we use EK_1 for encrypting a value which is passed as first input, EK_2 for the second input, and so on. In our construction, we utilize circuits with at most four distinct inputs. Therefore, FE.Setup is initialized with parameter 4. Occasionally, however, we require circuits with less inputs. This problem is not specifically addressed in [GGG+14]. One way to avoid this formality, and this is also implicitly assumed here, is to "stretch" every circuit to have four input wires in a way that it will ignore all which are not needed when evaluated. An alternative would be to instantiate different encryption key lists.

3.3 Security Properties

Since both sanitizable signatures and sanitizable signcryption share motivation, goal, and functionality, we can transfer (almost) all properties to our new primitive as well. Only minor modifications due to the slightly different structure are necessary. Since the definitions match the previous ones in the literature

closely, we give the concrete definitions for *(strong) unforgeability, immutability, (strong) privacy, strengthened unlinkability, and accountability* in Appendix A. Note that transparency is not listed here since we can argue that it is not achievable for any semi-oblivious sanitizable signcryption.

Furthermore, as we extend sanitizable signatures to signcryption schemes, we need to consider further confidentiality properties. Analogously to any encryption scheme, this boils down to derive a suitable notion of *indistinguishability*. Since the main goal of a semi-oblivious sanitizable signcryption scheme is to protect the classified data of the original document, both from the sanitizing party as well as from everyone not holding a valid decryption key and we gain a stronger version when defining indistinguishability against an adversary holding the sanitizer's key, we introduce the notion of $\mathsf{IND\text{-}CCA}_{\mathsf{san}}$ and remark that it implies security against the case that the adversary only has access to a decryption oracle SanDec compliant with the notion of IND-CCA security.

Since the malicious sanitizer can, with the help of its sanitization key, trivially compute sanitized cleartext versions of the challenge ciphertexts, we require that the message pairs forwarded to the left-or-right oracle only yield identical modifications on all admissible MOD to prevent trivial leakage. The same would be required if an outsider would have (only) access to a SanDec oracle.

Definition 3.3 *Let* $\mathsf{sOSSC} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{sig}}, \mathsf{KeyGen}_{\mathsf{san}}, \mathsf{SignEnc}, \mathsf{SanDec}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a semi-oblivious sanitizable signcryption scheme. We say that* sOSSC *is* $\mathsf{IND\text{-}CCA}_{\mathsf{san}}$-secure *if for all ppt adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{IND\text{-}CCA}_{\mathsf{san}}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of the $\mathsf{IND\text{-}CCA}_{\mathsf{san}}$ *experiment:*

$\mathsf{Exp}^{\mathsf{IND\text{-}CCA}_{\mathsf{san}}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda)$

$(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$;
$(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(\mathsf{Msk})$;
$(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(\mathsf{Msk})$;
$b \leftarrow \{0,1\}$;
$b' \leftarrow \mathcal{A}^{\mathsf{SignEnc}(\cdot,\cdot,sk_{\mathsf{sig}},\cdot,\mathsf{pparam}),\mathsf{LoREncSign}(\cdot,\cdot,\cdot,sk_{\mathsf{sig}},pk_{\mathsf{san}},b,\mathsf{pparam})}(pk_{\mathsf{sig}}, sk_{\mathsf{san}}, \mathsf{pparam})$;
if $b = b'$ **then**
  return 1.
else return 0.

$\mathsf{LoREncSign}((m_0, m_1), \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, b, \mathsf{pparam})$

if $|m_0| \neq |m_1|$ **then**
  return $\perp$.
else
  $c_{(m_b,\sigma_b)} \leftarrow \mathsf{SignEnc}(m_b, \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam})$;
  return $c_{(m_b,\sigma_b)}$.

We require that for any $((m_0, m_1), \mathrm{ADM})$ *queried to the* $\mathsf{LoREncSign}$ *oracle, we have that for all admissible modifications* MOD *with* $\mathrm{ADM}(\mathrm{MOD}) = 1$ *it holds that* $\mathrm{MOD}(m_0) = \mathrm{MOD}(m_1)$.

3.4 Security of Our Construction

Our generic construction naturally inherits most security properties from its underlying building blocks. This is summarized in the following proposition. Since the formal proof is a collection of standard reduction and its formal exhibition is rather lengthy and tedious, we will only provide some intuition after the formal statement and leave the full proof to be found in Appendix B.

We remark, however, that we need a strong requirement on the privacy of the underlying sanitizable schemes. That is, since security of the multi-input functional encryption scheme only holds as long as the available functions agree on the (partially replaced) answers of the challenge oracle ("compatibility"), we require that the output of the $\mathsf{LoREncSign}$ oracle in the $\mathsf{IND\text{-}CCA}_{\mathsf{san}}$ experiment does not split under the sanitizer's functionality. Since the output corresponds to sanitized pairs (m', σ') of either the left or right input, we need that the pairs agree in both cases. This is true by construction for m', because $\mathcal{A}$ can only make queries which yield the same modified message. It then follows for the signature part as well, if we presume perfect privacy of the sanitizable scheme, saying that sanitized signatures are independently distributed from the original message. An example of such a sanitizable scheme is the one by Brzuska et al. [BFLS10] based on group signatures, where the sanitizer signs the modified part with a fresh group signature.

Theorem 3.4 *Let* FE *be a* miIND*-secure multi-input functional encryption scheme and* $\mathcal{S}$ *be an unforgeable, immutable,* perfectly *private, unlinkable, non-interactive publicly accountable, and length-invariant sanitizable signature scheme. Then* sOSSC*, given in Construction 3.2, achieves unforgeability, immutability, perfect privacy, unlinkability, non-interactive public accountability, and* $\mathsf{IND\text{-}CCA}_{\mathsf{san}}$ *security. It does* not *achieve transparency.*

The idea of the proof is as follows. Since our scheme is a generic composition of a signature scheme and an encryption using the "sign-then-encrypt" approach, it is natural that most security properties for sanitizable schemes will remain intact. That is, the outer encryption scheme does not invalidate most properties of signature schemes. In addition, the security of the encryption scheme guarantees the indistinguishability notion (against the sanitizer, too), since functional encryption ensures that only the function's output —here, the sanitized version— is revealed. By stipulation, any challenge message pair, however, must yield the same sanitized versions, independently of the modification and due to perfect privacy of the underlying sanitizable scheme the adversary cannot deduce anything further about the challenge bit.

The only problem we might encounter is that the composition itself might reveal some information which should remain unknown. This is the case for transparency. Since, by definition of semi-oblivious sanitizable signcryption, any message-signature pair created by the signer will be encrypted, and any pair originating from the sanitizer will be in clear, we cannot hope to hide the information if a message was sanitized or not. All other properties, however, can be verified by a reduction to the corresponding property of the underlying sanitizable signature or multi-input functional encryption scheme.

4 (Fully) Oblivious Sanitizable Signcryption

The second scenario for sanitizable signcryption addresses the case when the sanitized pair should remain hidden from the sanitizer, too. Here, we will give the details of primitive, security properties, and our construction from fully homomorphic encryption.

This requires the message-signature pair to be encrypted until it reaches its final destination. In this section, we will give two approaches to build fully oblivious sanitizable signcryption. The first construction is an extension of Construction 3.2 for semi-oblivious sanitizable signcryption, where we update the

decryption key to re-encrypt the sanitized message under the receiver's key, while the second construction uses fully homomorphic encryption to operate directly on the ciphertext. Sadly, achieving security for the first construction is not straight-forwardly possible. We will elaborate on this further in the following.

There are two cases to consider here concerning the signer's a-priori knowledge about the receiver's identity. For this work, we only consider the setting where the identity of the potential receiver is known during signcryption, and that there is only a single receiver in the system. For the more general case, one needs to add some mechanisms that enable to specify legitimate receivers, say, via certified keys, or else the sanitizer itself may be able to recover the sanitized data by acting as a receiver.

4.1 The Primitive

In contrast to semi-oblivious sanitizable signcryption, we need to split sanitization from decryption and, hence, need two separate algorithms for handling these.

Definition 4.1 *An* oblivious sanitizable signcryption scheme *is a tuple of ppt algorithms* $\mathsf{OSSC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Signcrypt}, \mathsf{Sanit}, \mathsf{Unsigncrypt}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *such that*

Setup. *Upon input of the security parameter* 1^λ*, the algorithm outputs a master secret key* Msk *as well as public parameters* pparam*, i.e.,* $(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$.

Key Generation. *The algorithm* KeyGen *takes as input an index* $i \in \{\mathsf{sig}, \mathsf{san}, \mathsf{rec}\}$ *as well as the master secret key* Msk*, and outputs a pair of public and secret key, i.e.,* $(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(i, \mathsf{Msk})$.

Signcryption. *Upon input of a message* m*, a description* ADM*, a signer's secret key* sk_{sig}*, a sanitizer's public key* pk_{san}*, a receiver's public key* pk_{rcv}*, as well as public parameters* pparam*, the algorithm* $\mathsf{Signcrypt}$ *outputs an encrypted message-signature tuple (or* $\perp$ *in case of an error), i.e.,* $c_{(m,\sigma)} \leftarrow \mathsf{Signcrypt}(m, \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, pk_{\mathsf{rcv}}, \mathsf{pparam})$.

Sanitization. *Upon input of an encrypted message-signature pair* $c_{(m,\sigma)}$*, a description of desired modifications* MOD*, a sanitizer's secret key* sk_{san}*, a signer's public key* pk_{sig}*, a receiver's public key* pk_{rcv}*, and public parameters* pparam*, the algorithm* Sanit *outputs a sanitized and still encrypted version of the message-signature pair (or* $\perp$ *in case of an error), i.e.,* $c_{(m',\sigma')} \leftarrow \mathsf{Sanit}(c_{(m,\sigma)}, \mathrm{MOD}, sk_{\mathsf{san}}, pk_{\mathsf{sig}}, pk_{\mathsf{rcv}}, \mathsf{pparam})$.

Unsigncryption. *Upon input of an encrypted message-signature pair* $c_{(m,\sigma)}$*, a receiver's secret key* sk_{rcv}*, public keys of both signer* pk_{sig} *and sanitizer* pk_{san}*, and public parameters* pparam*, the algorithm* $\mathsf{Unsigncrypt}$ *outputs a decrypted and sanitized version of the message-signature pair if it verifies under the given public keys (or* $\perp$ *in case of an error), i.e.,* $(m, \sigma) \leftarrow \mathsf{Unsigncrypt}(c_{(m,\sigma)}, sk_{\mathsf{rcv}}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam})$.

Verify. *Upon input of a (possibly encrypted) message-signature pair* $\widetilde{\mathsf{m}\sigma}$*, public keys of both signer* pk_{sig} *and sanitizer* pk_{san}*, as well as public parameters* pparam*, the algorithm* Verify *returns a bit* b *indicating whether the (encrypted) message-signature pair is valid or not. We require* Verify *to be deterministic and, hence, write* $\mathsf{Verify}(\widetilde{\mathsf{m}\sigma}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam}) = b$.

Proof. *Upon input of a signer's secret key* sk_{sig}*, a (possibly encrypted) message-signature pair* $\widetilde{\mathsf{m}\sigma}$*, a possibly empty list of (encrypted) message-signature pairs* $\{\widetilde{\mathsf{m}_1\sigma_1}, \ldots, \widetilde{\mathsf{m}_t\sigma_t}\}$*, a sanitizer's public key* pk_{san}*, and public parameters* pparam*, the algorithm* Proof *outputs a string* $\pi \in \{0,1\}^* \cup \{\perp\}$*, i.e.,* $\pi \leftarrow \mathsf{Proof}(sk_{\mathsf{sig}}, \widetilde{\mathsf{m}\sigma}, \{\widetilde{\mathsf{m}_1\sigma_1}, \ldots, \widetilde{\mathsf{m}_t\sigma_t}\}, pk_{\mathsf{san}}, \mathsf{pparam})$.

Judge. *Upon input of a (possibly encrypted) message-signature pair* $\widetilde{\mathsf{m}\sigma}$*, public keys of both signer* pk_{sig} *and sanitizer* pk_{san}*, a string* $\pi \in \{0,1\}^* \cup \{\perp\}$*, as well as public parameters* pparam*, the algorithm* Judge *outputs a bit* d *indicating which party has generated the given pair (or* $\perp$ *in case of an error). Here,* $d = 1$ *refers to the signer and* $d = 0$ *to the sanitizer. We require* Judge *to be deterministic and write* $\mathsf{Judge}((\widetilde{m}, \widetilde{\sigma}), pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \pi, \mathsf{pparam}) = d$*.*

We expect a oblivious sanitizable signcryption scheme to have the following correctness properties:

Perfect correctness. For all security parameters $\lambda \in \mathbb{N}$, all system instantiations $(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$, all signer's key pairs $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}(\mathsf{sig}, \mathsf{Msk})$, all sanitizer's key pairs $(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}(\mathsf{san}, \mathsf{Msk})$, all receiver's key pairs $(pk_{\mathsf{rcv}}, sk_{\mathsf{rcv}}) \leftarrow \mathsf{KeyGen}(\mathsf{rec}, \mathsf{Msk})$, all messages $m \in \mathcal{M}$, all descriptions $\text{ADM} \in \{0,1\}^*$, all encrypted signatures $c_{(m,\sigma)} \leftarrow \mathsf{Signcrypt}(m, \text{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, pk_{\mathsf{rcv}}, \mathsf{pparam})$, all descriptions $\text{MOD} \in \{0,1\}^*$ which match ADM, all encrypted derived signatures $c_{(m',\sigma')} \leftarrow \mathsf{Sanit}(c_{(m,\sigma)}, \text{MOD}, sk_{\mathsf{san}}, pk_{\mathsf{sig}}, pk_{\mathsf{rcv}}, \mathsf{pparam})$ with $\text{MOD}(m) = m'$, and all decrypted tuple $(m', \sigma') \leftarrow \mathsf{Unsigncrypt}(c_{(m',\sigma')}, sk_{\mathsf{rcv}}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam})$ we have

$$\mathsf{Verify}((m', \sigma'), pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam}) = 1 \ .$$

Perfect proof correctness. For all security parameters $\lambda \in \mathbb{N}$, all system instantiations $(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$, all signer's key pairs $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}(\mathsf{sig}, \mathsf{Msk})$, all sanitizer's key pairs $(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}(\mathsf{san}, \mathsf{Msk})$, all receiver's key pairs $(pk_{\mathsf{rcv}}, sk_{\mathsf{rcv}}) \leftarrow \mathsf{KeyGen}(\mathsf{rec}, \mathsf{Msk})$, all messages $m \in \mathcal{M}$, all descriptions $\text{ADM} \in \{0,1\}^*$, all encrypted signatures $c_{(m,\sigma)} \leftarrow \mathsf{Signcrypt}(m, \text{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, pk_{\mathsf{rcv}}, \mathsf{pparam})$, all descriptions $\text{MOD} \in \{0,1\}^*$ which match ADM, all encrypted derived signatures $c_{(m',\sigma')} \leftarrow \mathsf{Sanit}(c_{(m,\sigma)}, \text{MOD}, sk_{\mathsf{san}}, pk_{\mathsf{sig}}, pk_{\mathsf{rcv}}, \mathsf{pparam})$ where $\text{MOD}(m) = m'$, all decrypted tuple $(m', \sigma') \leftarrow \mathsf{Unsigncrypt}(c_{(m',\sigma')}, sk_{\mathsf{rcv}}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam})$, any (polynomially many) valid, possibly encrypted, message-signature pairs $\widetilde{\mathsf{m}_1\sigma_1}, \ldots, \widetilde{\mathsf{m}_t\sigma_t}$ such that $c_{(m,\sigma)} = \widetilde{\mathsf{m}_i\sigma_i}$ holds for some $i \in \{1, \ldots, t\}$, any any proof $\pi \leftarrow \mathsf{Proof}(sk_{\mathsf{sig}}, (m', \sigma'), \{\widetilde{\mathsf{m}_1\sigma_1}, \ldots, \widetilde{\mathsf{m}_t\sigma_t}\}, pk_{\mathsf{san}}, \mathsf{pparam})$ we have

$$\mathsf{Judge}((m', \sigma'), pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \pi, \mathsf{pparam}) = 0 \ .$$

4.2 Security Properties

The basic security properties remain unchanged, only that the adversary now gets the receiver's secret key as additional input. Recall that the basic properties of sanitizable schemes, such as privacy of the original data, should still hold against malicious receivers.

As for confidentiality, we now aim at hiding any information about the sanitized message, too. To this end, starting with the semi-oblivious sanitizable signcryption experiment $\mathsf{IND\text{-}CCA}_{\mathsf{san}}$, we relax the requirement on the queries to the challenge oracle $\mathsf{LoREncSign}$ that $\text{MOD}(m_0) = \text{MOD}(m_1)$ for any admissible modification. Instead we now only require that $|\text{MOD}(m_0)| = |\text{MOD}(m_1)|$ for any admissible modification (such that the adversary cannot infer anything about the original message from the ciphertext length). We further augment the $\mathsf{IND\text{-}CCA}_{\mathsf{san}}$ scenario by also giving the adversary access to an $\mathsf{Unsigncrypt}$ oracle now, allowing the adversary to ask for decryptions at will. There is only one restriction: if the adversary asks $\mathsf{Unsigncrypt}$ about a ciphertext returned by the challenge oracle $\mathsf{LoREncSign}$, then the corresponding query must again have $\text{MOD}(m_0) = \text{MOD}(m_1)$ for any admissible modification. We could not pose this restriction if we gave the adversary the receiver's secret key here. The above somehow resurrects the $\mathsf{IND\text{-}CCA}_{\mathsf{san}}$ security and, at the same time, prevents trivial attacks. We call the security property where an adversary can ask for arbitrary decryptions, except for challenge ciphertexts, of course, $\mathsf{IND\text{-}CCA}_{\mathsf{rcv}}$ as confidentiality should be provided all the way to the receiver. If we drop the $\mathsf{Unsigncrypt}$ oracle, we speak of $\mathsf{IND\text{-}CPA}_{\mathsf{rcv}}$ security.

Definition 4.2 *Let* $\mathsf{OSSC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Signcrypt}, \mathsf{Sanit}, \mathsf{Unsigncrypt}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *be a oblivious sanitizable signcryption scheme. We say that* OSSC *is* $\mathsf{IND\text{-}CCA}_{\mathsf{rcv}}$-secure *if for all ppt adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{IND\text{-}CCA}_{\mathsf{rcv}}}_{\mathcal{A},\mathsf{OSSC}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins $\mathcal{A}$ *and the* $\mathsf{IND\text{-}CCA}_{\mathsf{rcv}}$ *experiment below:*

$\mathsf{Exp}^{\mathsf{IND\text{-}CCA}_{\mathsf{rcv}}}_{\mathcal{A},\mathsf{OSSC}}(\lambda)$

$(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$;
$(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}(\mathsf{sig}, \mathsf{Msk})$;
$(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}(\mathsf{san}, \mathsf{Msk})$;
$(pk_{\mathsf{rcv}}, sk_{\mathsf{rcv}}) \leftarrow \mathsf{KeyGen}(\mathsf{rcv}, \mathsf{Msk})$;
$b \leftarrow \{0,1\}$;
$b' \leftarrow \mathcal{A}^{\mathsf{Signcrypt}(\cdot,\cdot,sk_{\mathsf{sig}},pk_{\mathsf{san}},pk_{\mathsf{rcv}},\mathsf{pparam}),\mathsf{LoREncSign}(\cdot,\cdot,sk_{\mathsf{sig}},pk_{\mathsf{san}},pk_{\mathsf{rcv}},b,\mathsf{pparam}),\mathsf{Unsigncrypt}(\cdot,sk_{\mathsf{rcv}},pk_{\mathsf{sig}},pk_{\mathsf{san}},\mathsf{pparam})}(pk_{\mathsf{sig}}, sk_{\mathsf{san}}, \mathsf{pparam})$;
if $b = b'$ **then**
  return 1.
else return 0.

$\mathsf{LoREncSign}((m_0, m_1), \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, pk_{\mathsf{rcv}}, b, \mathsf{pparam})$

if $|m_0| \neq |m_1| \vee |\mathrm{MOD}(m_0)| \neq |\mathrm{MOD}(m_1)|$ *for any* MOD *with* $\mathrm{ADM}(\mathrm{MOD}) = 1$ **then**
  return $\perp$.
else
  $c_{(m_b,\sigma_b)} \leftarrow \mathsf{Signcrypt}(m_b, \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, pk_{\mathsf{rcv}}, \mathsf{pparam})$;
  return $c_{(m_b,\sigma_b)}$.

We require that for any $c_{(m,\sigma)}$ *returned by the* LoREncSign *oracle upon query* $((m_0, m_1), \mathrm{ADM})$, $\mathcal{A}$ *only askes the* Unsigncrypt *oracle for a decryption of* $c_{(m,\sigma)}$ *if for all admissible modifications* MOD *with* $\mathrm{ADM}(\mathrm{MOD}) = 1$ *it holds that* $\mathrm{MOD}(m_0) = \mathrm{MOD}(m_1)$.

If no Unsigncrypt *oracle exists we obtain the corresponding* $\mathsf{IND\text{-}CPA}_{\mathsf{rcv}}$*-security notion.*

4.3 On the Hardness of Extending Construction 3.2

Let us first explain why we cannot achieve $\mathsf{IND\text{-}CCA}_{\mathsf{rcv}}$-security by simply extending Construction 3.2. Assume that, instead of holding a key $\mathsf{sk}_{\mathsf{Sanit}}$ for the $\mathsf{S.Sanit}(\cdot,\cdot,sk_{\mathsf{san}},\cdot)$ circuit, the sanitizer will now be given a key for the

$$\mathsf{sk}_{\mathsf{san}} = \mathsf{Enc}(\cdot, \mathsf{S.Sanit}(\cdot,\cdot,sk_{\mathsf{san}},\cdot))$$

circuit which eventually encrypts the result under the receiver's key using any traditional public key encryption scheme. Intuitively, decrypting a ciphertext with the above multi-input functional decryption key will result in evaluating the above circuit upon the underlying plaintexts.

While this construction is functionally sound, we cannot transfer the security proof, not even to the $\mathsf{IND\text{-}CPA}_{\mathsf{rcv}}$ setting. The reason is that in the proof for the semi-oblivious sanitizable signcryption case we have used the fact that the adversary's queries to the LoREncSign challenge oracle satisfy $\mathrm{MOD}(m_0) = \mathrm{MOD}(m_1)$ for all admissible modifications. In combination with perfect privacy of the sanitizable signature scheme, this guaranteed the necessary compatibility for the reduction to the miIND-security, saying that

the messages in the challenge queries cannot be split by the santizer's key $\mathsf{sk}_{\mathsf{Sanit}}$. Here, in the $\mathsf{IND\text{-}CPA}_{\mathsf{rcv}}$ setting, the adversary may now ask arbitrary messages with different sanitized versions, such that we cannot reproduce the reduction. More concretely, for reproducing the reduction we would need to have that all sanitation outputs for any combinations of inputs yield the same result independent of whether m_0 or m_1 was chosen. As for the semi-oblivious case, we could add the extra requirement that $\text{MOD}(m_0) = \text{MOD}(m_1)$ holds for any admissible modification and, hence, gain a notion of indistinguishability given perfect privacy, however, since our goal is to protect the sanitized message it would not be very useful to utilize a security definition which de facto only permits equal sanitizations.

4.4 A Construction from FHE

Now, we present a construction from fully homomorphic encryption.

Construction 4.3 *Assume a sanitizable signature scheme* $\mathcal{S} = (\mathsf{S.KeyGen}_{\mathsf{sig}}, \mathsf{S.KeyGen}_{\mathsf{san}}, \mathsf{S.Sign}, \mathsf{S.Sanit}, \mathsf{S.Verify}, \mathsf{S.Proof}, \mathsf{S.Judge})$ *and a fully homomorphic encryption scheme* $\mathcal{FHE} = (\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$. *Then we define* $\mathsf{OSSC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Signcrypt}, \mathsf{Sanit}, \mathsf{Unsigncrypt}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *as follows.*

Setup. *Since this algorithm is not needed for this construction the procedure returns* $\perp$.

$\mathsf{Setup}(1^\lambda)$:

$\mathsf{Msk} \leftarrow \perp$;
$\mathsf{pparam} \leftarrow \perp$;
return $(\mathsf{Msk}, \mathsf{pparam})$.

Key Generation. *The algorithm* KeyGen *takes as input an index* $i \in \{\mathsf{sig}, \mathsf{san}, \mathsf{rec}\}$ *as well as the master secret key* Msk. *If* $i = \mathsf{sig}$, *it generates a signer's key pair* $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}})$ *for* $\mathcal{S}$ *and outputs it. If* $i = \mathsf{san}$, *generates a sanitizer's key pair for* $\mathcal{S}$ *and outputs it. Finally, if* $i = \mathsf{rec}$, *it generates a key pair* $(sk_{\mathsf{rcv}}, pk_{\mathsf{rcv}})$ *for* $\mathcal{FHE}$ *and outputs it.*

$\mathsf{KeyGen}(i, \mathsf{Msk})$:

if $i == \mathsf{sig}$ **then**
  $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{S.KeyGen}_{\mathsf{sig}}(1^\lambda)$;
  return $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}})$.
else if $i == \mathsf{san}$ **then**
  $(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{S.KeyGen}_{\mathsf{san}}(1^\lambda)$;
  return $(pk_{\mathsf{san}}, sk_{\mathsf{san}})$.
else if $i == \mathsf{rec}$ **then**
  $(pk_{\mathsf{rcv}}, sk_{\mathsf{rcv}}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$;
  return $(pk_{\mathsf{rcv}}, sk_{\mathsf{rcv}})$.
else return $\perp$.

Signcryption. *Upon input of a message* m, *a description* ADM *of admissible parts, a signer's secret key* sk_{sig}, *a sanitizer's public key* pk_{san}, *a receiver's public key* pk_{rcv}, *as well as public parameters* pparam, *the algorithm* Signcrypt *uses the sanitizable signature scheme to compute a signature* σ *on* m *with* ADM *under* sk_{sig} *and* pk_{san}, *encrypts both message and signature to* $c_{(m,\sigma)}$ *under* pk_{rcv}, *and returns this value (or* $\perp$ *in case of an error).*

$\underline{\mathsf{Signcrypt}(m, \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, pk_{\mathsf{rcv}}, \mathsf{pparam}) :}$

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

$c_{(m,\sigma)} \leftarrow \mathsf{FHE.Enc}(pk_{\mathsf{rcv}}, (m, \sigma));$

return $c_{(m,\sigma)}$.

Sanitization. *Upon input of an encrypted message-signature pair $c_{(m,\sigma)}$, a description of desired modifications* MOD*, a sanitizer's secret key sk_{san}, a signer's public key pk_{sig}, a receiver's public key pk_{rcv}, and public parameters* pparam*, the algorithm* Sanit *encrypts pk_{sig} and* MOD *under pk_{rcv}, runs* FHE.Eval *on the* S.Sanit *circuit with inputs $c_{(m,\sigma)}, c_{pk_{\mathsf{sig}}}$, and c_{MOD} to get $c_{(m',\sigma')}$, which it outputs.*

$\underline{\mathsf{Sanit}(c_{(m,\sigma)}, \mathrm{MOD}, sk_{\mathsf{san}}, pk_{\mathsf{sig}}, pk_{\mathsf{rcv}}, \mathsf{pparam}) :}$

$c_{pk_{\mathsf{sig}}} \leftarrow \mathsf{FHE.Enc}(pk_{\mathsf{rcv}}, pk_{\mathsf{sig}});$

$c_{\mathrm{MOD}} \leftarrow \mathsf{FHE.Enc}(pk_{\mathsf{rcv}}, \mathrm{MOD});$

$c_{(m',\sigma')} \leftarrow \mathsf{FHE.Eval}(\mathsf{S.Sanit}(\cdot, \cdot, sk_{\mathsf{san}}, \cdot), c_{(m,\sigma)}, c_{pk_{\mathsf{sig}}}, c_{\mathrm{MOD}});$

return $c_{(m',\sigma')}$.

Unsigncryption. *Upon input of an encrypted message-signature pair $c_{(m,\sigma)}$, a receiver's secret key sk_{rcv}, public keys of both signer pk_{sig} and sanitizer pk_{san}, and public parameters* pparam*, the algorithm* Unsigncrypt *decrypts the pair using* FHE.Dec *with sk_{rcv}, then runs* S.Verify *upon the decrypted values, and outputs them if the signature was found valid (or $\perp$ in case of an error).*

$\underline{\mathsf{Unsigncrypt}(c_{(m,\sigma)}, sk_{\mathsf{rcv}}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam}) :}$

$(m, \sigma) \leftarrow \mathsf{FHE.Dec}(sk_{\mathsf{rcv}}, c_{(m,\sigma)});$

if $\mathsf{S.Verify}((m, \sigma), pk_{\mathsf{sig}}, pk_{\mathsf{san}}) == 1$ **then**

  return (m, σ).

else return $\perp$.

Verify. *Upon input of a (encrypted) message-signature pair $\widetilde{\mathsf{m}\sigma}$, public keys of both signer pk_{sig} and sanitizer pk_{san}, as well as public parameters* pparam*, the algorithm* Verify *outputs the value* S.Verify *returns which is assumed to return $\perp$ upon an encrypted input[4].*

$\underline{\mathsf{Verify}(\widetilde{\mathsf{m}\sigma}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam}) :}$

return $\mathsf{S.Verify}((m, \sigma), pk_{\mathsf{sig}}, pk_{\mathsf{san}})$.

Proof. *The algorithm* Proof *always outputs an empty proof $\perp$.*

$\underline{\mathsf{Proof}(sk_{\mathsf{sig}}, \widetilde{\mathsf{m}\sigma}, \{\widetilde{\mathsf{m}_1\sigma_1}, \ldots, \widetilde{\mathsf{m}_t\sigma_t}\}, pk_{\mathsf{san}}, \mathsf{pparam}) :}$

return $\perp$.

Judge. *Upon input of a (encrypted) message-signature pair $\widetilde{\mathsf{m}\sigma}$, public keys of both signer pk_{sig} and sanitizer pk_{san}, a string $\pi \in \{0,1\}^* \cup \{\perp\}$, as well as public parameters* pparam*, the algorithm* Judge *outputs the value of* S.Judge *which is assumed to return $\perp$ upon an encrypted input.*

$\underline{\mathsf{Judge}(\widetilde{\mathsf{m}\sigma}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \pi, \mathsf{pparam}) :}$

return $\mathsf{S.Judge}((m, \sigma), pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \pi)$.

Note that achieving $\mathsf{IND\text{-}CCA}_{\mathsf{rcv}}$ security with a decryption oracle under the receiver's key is infeasible, as the FHE schemes usually do not achieve full chosen-ciphertext security. This is why we revert to $\mathsf{IND\text{-}CPA}_{\mathsf{rcv}}$ security For showing the other properties of sanitizable schemes it suffices to prove them when holding the decryption key of the FHE scheme.

Theorem 4.4 *Let $\mathcal{FHE}$ be an* IND-CPA*-secure fully homomorphic encryption scheme, and $\mathcal{S}$ be an unforgeable, immutable, private, unlinkable, non-interactive publicly accountable, and length-invariant sanitizable signature scheme. Then the oblivious sanitizable signcryption scheme given in Construction 4.3 achieves unforgeability, immutability, privacy, unlinkability, non-interactive public accountability, and* $\mathsf{IND\text{-}CPA}_{\mathsf{rcv}}$ *security. It does not achieve transparency.*

The proof of the $\mathsf{IND\text{-}CPA}_{\mathsf{rcv}}$ property is straightforward, and follows from a reduction to the IND-CPA security of the underlying fully homomorphic encryption scheme, analogously to the case for multi-input functional encryption. For the other properties, note that playing against the underlying sanitizable signature scheme, the reduction would generate the keys for the FHE scheme and can thus always "peel off" the outer encryption and interact with the corresponding interfaces of the sanitizable schemes.

It is easy to see that Construction 4.3 cannot achieve transparency. Unlike for the semi-oblivious sanitizable signcryption case, the reason does lie within the structure of the primitive but rather in the nature of its security properties. Since we aim to achieve non-interactive public accountability any party, thus also the adversary holding the receiver's secret key, can evaluate the Judge algorithm and directly learn which party is accountable, hence, if the tuple was sanitized or not. One option to circumvent this trivial distinction is to disallow the adversary to decrypt, i.e., to deny him access to the receiver's secret key, and thereby restricting the property. If the underlying fully homomorphic encryption scheme is circuit private then it implicitly hides whether an evaluation, in our case sanitation, has taken place. Nevertheless, we omit this weaker version since we wish to achieve insider[5] security notions for our primitive to capture malicious receivers who might very well have the highest interest in knowing whether or not they face a modified document or the original one.

5 Conclusion

We have raised the issue of sanitization over encrypted data and given a feasibility result that it is possible to build such signature schemes. In the case of oblivious sanitizable signcryption schemes and strong security guarantees, it remains open to derive schemes which provide $\mathsf{IND\text{-}CCA}_{\mathsf{rcv}}$-security, ideally together with public verifiability. Even our $\mathsf{IND\text{-}CPA}_{\mathsf{rcv}}$-solution based on FHE does not achieve public verifiability.

Despite the theoretical appeal, the involved primitives in our constructions are far from being practical. Even more, it remains to be shown that an efficient multi-input functional encryption scheme exists. So far, all existing constructions rely on obfuscation and non-interactive zero-knowledge proofs for all efficiently computable (deterministic) functions [GGG+14], multi-linear maps [BLR+14], or weaker assumptions under certain restrictions [BKS15]. One direction would be to come up with an efficient functional encryption scheme which would match our needs. Alternatively, one can try to solve the problem in question directly, with a tailored, non-generic solution to satisfy sanitizable signcryption. In any case, achieving a practical solution remains an interesting open problem. Here, signcryption with its original intent to combine encryption and signatures with improved efficiency and functionality may be a promising direction to follow.

[5]Insider-version refers to the fact that also entities inside the system may be regarded as adversaries, like sanitizers in our case.

For some applications of fully oblivious sanitizable signcryption, it may be the case that the receiver is not known at signature creation.[6] This is currently not covered by the solution based on fully homomorphic encryption. For specifying a receiver later on, a potential fix is to use a *key-switching mechanism* [BV11, CCL+14] which allows for a user, here the signer, to derive a so-called switch-key using his secret key, allowing any party to change a ciphertext under his own public key to an encryption under the public key specified in the switch-key. This remains yet to be explored.

Acknowledgments

This work has been funded by the DFG as part of project P2 within the CRC 1119 CROSSING. Marc Fischlin is supported by the Heisenberg grant Fi 940/3-2 of the German Research Foundation (DFG). This work was done in part while Marc was visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

References

[AAB+13] Shashank Agrawal, Shweta Agrawal, Saikrishna Badrinarayanan, Abishek Kumarasubramanian, Manoj Prabhakaran, and Amit Sahai. Function private functional encryption and property preserving encryption : New definitions and positive results. Cryptology ePrint Archive, Report 2013/744, 2013. `http://eprint.iacr.org/`. (Cited on page 8.)

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

[ACdT05] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS 2005: 10th European Symposium on Research in Computer Security*, volume 3679 of *Lecture Notes in Computer Science*, pages 159–177, Milan, Italy, September 12–14, 2005. Springer, Berlin, Germany. (Cited on pages 1, 5, and 6.)

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. Cryptology ePrint Archive, Report 2015/173, 2015. `http://eprint.iacr.org/`. (Cited on page 2.)

[BDF+14] Michael Backes, Ozgur Dagdelen, Marc Fischlin, Sebastian Gajek, Sebastian Meiser, and Dominique Schröder. Operational signature schemes. Cryptology ePrint Archive, Report 2014/820, 2014. `http://eprint.iacr.org/2014/820`. (Cited on page 3.)

[BF14] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 520–537, Buenos Aires, Argentina, March 26–28, 2014. Springer, Berlin, Germany. (Cited on page 3.)

[6] As discussed, there are still interesting cases where the receiver may be known in advance but where the possibility of "on-the-fly" sanitization may still be desirable.

[BFF+09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 317–336, Irvine, CA, USA, March 18–20, 2009. Springer, Berlin, Germany. (Cited on pages 1 and 6.)

[BFLS10] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 444–461, Paris, France, May 26–28, 2010. Springer, Berlin, Germany. (Cited on pages 1, 3, 6, and 16.)

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. Springer, Berlin, Germany. (Cited on page 3.)

[BKS15] Zvika Brakerski, Ilan Komargodski, and Gil Segev. From single-input to multi-input functional encryption in the private-key setting. Cryptology ePrint Archive, Report 2015/158, 2015. `http://eprint.iacr.org/`. (Cited on pages 2 and 22.)

[BLR+14] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/834, 2014. `http://eprint.iacr.org/`. (Cited on pages 2 and 22.)

[BMS13] Michael Backes, Sebastian Meiser, and Dominique Schröder. Delegatable functional signatures. Cryptology ePrint Archive, Report 2013/408, 2013. `http://eprint.iacr.org/2013/408`. (Cited on page 3.)

[BPS12] Christina Brzuska, Henrich C. Pöhls, and Kai Samelin. Non-interactive public accountability for sanitizable signatures. In *Proc. of the 9th European PKI Workshop: Research and Applications (EuroPKI 2012)*, volume 7868 of *Lecture Notes in Computer Science (LNCS)*, page 178. Springer-Verlag, 2012. This is an extended and revised version of the original publication. (Cited on pages 3, 5, and 14.)

[BRS13] Dan Boneh, Ananth Raghunathan, and Gil Segev. Function-private identity-based encryption: Hiding the function in functional encryption. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 461–478. Springer Berlin Heidelberg, 2013. (Cited on page 8.)

[BS15] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. In Yevgeniy Dodis and JesperBuus Nielsen, editors, *Theory of Cryptography*, volume 9015 of *Lecture Notes in Computer Science*, pages 306–324. Springer Berlin Heidelberg, 2015. (Cited on page 8.)

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273, Providence, RI, USA, March 28–30, 2011. Springer, Berlin, Germany. (Cited on page 2.)

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Palm Springs, California, USA, October 22–25, 2011. IEEE Computer Society Press. (Cited on page 23.)

[CCL+14] Nishanth Chandran, Melissa Chase, Feng-Hao Liu, Ryo Nishimaki, and Keita Xagawa. Re-encryption, functional re-encryption, and multi-hop re-encryption: A framework for achieving obfuscation-based security and instantiations from lattices. In Hugo Krawczyk, editor, *Public-Key Cryptography– PKC 2014*, volume 8383 of *Lecture Notes in Computer Science*, pages 95–112. Springer Berlin Heidelberg, 2014. (Cited on page 23.)

[CJL12] Sébastien Canard, Amandine Jambert, and Roch Lescuyer. Sanitizable signatures with several signers and sanitizers. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *AFRICACRYPT 12: 5th International Conference on Cryptology in Africa*, volume 7374 of *Lecture Notes in Computer Science*, pages 35–52, Ifrance, Morocco, July 10–12, 2012. Springer, Berlin, Germany. (Cited on page 3.)

[CLM08] Sébastien Canard, Fabien Laguillaumie, and Michel Milhau. Trapdoor sanitizable signatures and their application to content protection. In StevenM. Bellovin, Rosario Gennaro, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, volume 5037 of *Lecture Notes in Computer Science*, pages 258–276. Springer Berlin Heidelberg, 2008. (Cited on pages 1 and 3.)

[CLTV14] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. Cryptology ePrint Archive, Report 2014/882, 2014. `http://eprint.iacr.org/`. (Cited on page 11.)

[DZ10] Alexander W. Dent and Yuliang Zheng. *Practical Signcryption*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010. (Cited on page 3.)

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, Maryland, USA, May 31 – June 2, 2009. ACM Press. (Cited on pages 3 and 9.)

[GGG+14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *EUROCRYPT*, pages 578–602. Springer, 2014. (Cited on pages 2, 6, 7, 8, 14, and 22.)

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. Cryptology ePrint Archive, Report 2013/451, 2013. `http://eprint.iacr.org/`. (Cited on page 7.)

[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 Masayuki Abe and Virgil Gligor, editors, *ASIACCS 08: 3rd Conference on Computer and Communications Security*, pages 353–362, Tokyo, Japan, March 18–20, 2008. ACM Press. (Cited on page 1.)

[IKO+07] Tetsuya Izu, Noboru Kunihiro, Kazuo Ohta, Masahiko Takenaka, and Takashi Yoshioka. A sanitizable signature scheme with aggregation. In Ed Dawson and Duncan S. Wong, editors, *Information Security Practice and Experience*, volume 4464 of *Lecture Notes in Computer Science*, pages 51–64. Springer Berlin Heidelberg, 2007. (Cited on page 3.)

[JMSW02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 244–262, San Jose, CA, USA, February 18–22, 2002. Springer, Berlin, Germany. (Cited on page 1.)

[Mey13] David Meyer. Nokia: Yes, we decrypt your https data, but don't worry about it, January 2013. `https://gigaom.com/2013/01/10/nokia-yes-we-decrypt-your-https-data-but-dont-worry-about-it/`. (Cited on page 2.)

[ML02] John Malone-Lee. Identity-based signcryption. Cryptology ePrint Archive, Report 2002/098, 2002. `http://eprint.iacr.org/`. (Cited on page 3.)

[MSI+03] K Miyazaki, S Susaki, M Iwamura, T Matsumoto, R Sasaki, and H Yoshiura. Digital documents sanitizing problem. *IEICE*, (195):61–67, 2003. (Cited on page 1.)

[MSP10] Yang Ming, Xiaoqin Shen, and Yamian Peng. Identity-based sanitizable signature scheme in the standard model. In Rongbo Zhu, Yanchun Zhang, Baoxiang Liu, and Chunfeng Liu, editors, *Information Computing and Applications*, volume 105 of *Communications in Computer and Information Science*, pages 9–16. Springer Berlin Heidelberg, 2010. (Cited on page 3.)

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. `http://eprint.iacr.org/`. (Cited on page 2.)

[SBZ02] Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content extraction signatures. In Kwangjo Kim, editor, *ICISC 01: 4th International Conference on Information Security and Cryptology*, volume 2288 of *Lecture Notes in Computer Science*, pages 285–304, Seoul, Korea, December 6–7, 2002. Springer, Berlin, Germany. (Cited on page 1.)

[SSW09] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 457–473. Springer, Berlin, Germany, March 15–17, 2009. (Cited on page 8.)

[SW08] Amit Sahai and Brent Waters. Slides on functional encryption. PowerPoint presentation, 2008. `http://www.cs.utexas.edu/~bwaters/presentations/files/functional.ppt`. (Cited on page 2.)

[YSL10] Dae Hyun Yum, Jae Woo Seo, and Pil Joong Lee. Trapdoor sanitizable signatures made easy. In Jianying Zhou and Moti Yung, editors, *ACNS 10: 8th International Conference on Applied Cryptography and Network Security*, volume 6123 of *Lecture Notes in Computer Science*, pages 53–68, Beijing, China, June 22–25, 2010. Springer, Berlin, Germany. (Cited on page 3.)

[Zhe97] Yuliang Zheng. Digital signcryption or how to achieve cost(signature & encryption) $\ll$ cost(signature) + cost(encryption). In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 165–179, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Berlin, Germany. (Cited on page 3.)

A Security Properties for Semi-Oblivious Sanitizable Signcryption

Here, we will formally describe the security notions for semi-oblivious sanitizable signcryption which can be inherited from sanitizable signatures, namely, (strong) unforgeability, immutability, (strong) privacy, strengthened unlinkability, and non-interactive public accountability.

The most basic notion every signature or signcryption scheme should fulfill is unforgeability. Intuitively, it requires that nobody should be able to derive a valid message-signature pair without knowledge of the secret key. In our case, this includes knowledge of either the signer's secret key sk_{sig} or the sanitizer's secret key sk_{san} (the later covered by immutability). Note that a signature is a forgery as soon as the Verify algorithm accepts the input. In contrast to normal signatures, our semi-oblivious sanitizable signcryption scheme gains an extra feature, namely, that a message-signature pair can be valid either in encrypted or decrypted form. Since the Verify algorithm works for both kinds of inputs, a forgery can also either be either an encrypted or decrypted pair.

Definition A.1 *A semi-oblivious sanitizable signcryption scheme* $\mathsf{sOSSC} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{sig}}, \mathsf{KeyGen}_{\mathsf{san}}, \mathsf{SignEnc}, \mathsf{SanDec}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *is* strongly unforgeable *if for every ppt adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{SUnforgeability}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda) \ ,$$

where the probability is taken over the random coins of $\mathcal{A}$ *and the* SUnforgeability *experiment which is defined as follows:*

$\mathsf{Exp}^{\mathsf{SUnforgeability}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda)$

$(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$;
$(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(\mathsf{Msk})$;
$(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(\mathsf{Msk})$;
$\widetilde{\mathsf{m}^*\sigma^*} \leftarrow \mathcal{A}^{\mathsf{SignEnc}(\cdot,\cdot,sk_{\mathsf{sig}},\cdot,\mathsf{pparam}),\mathsf{SanDec}(\cdot,\cdot,sk_{\mathsf{san}},\cdot,\mathsf{pparam}),\mathsf{Proof}(sk_{\mathsf{sig}},\cdot,\cdot,\cdot)}(pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam})$;
if $(\mathsf{Verify}(\widetilde{\mathsf{m}^*\sigma^*}, pk_{\mathsf{sig}}, pk_{\mathsf{san}}) = 1$
$\wedge$ "$\mathsf{SignEnc}(\cdot,\cdot,sk_{\mathsf{sig}},\cdot,\mathsf{pparam})$ *was not queried for* $m^*, \mathrm{ADM}^*, pk_{\mathsf{san}}$ *where* ADM^* *is recoverable from* σ^*"
$\wedge$ "$\mathsf{SanDec}(c_{(m,\sigma)}, \mathrm{MOD}, pk_{\mathsf{sig}}, sk_{\mathsf{san}}, \mathsf{pparam})$ *did not return* $\widetilde{\mathsf{m}^*\sigma^*}$ *for any query*"
return 1.
else return 0.

Recall that the notion of immutability intuitively says that it should not be feasible to modify blocks which are not designated in the description of admissible parts. In other words, malicious sanitizers should not be able to change unintended portions of the message and still derive a valid signature. We stress again, that the inability to derive valid message-signature pairs in a manner outside of what is allowed by ADM also means and implies, that a corrupted sanitizer can also not forge any signatures which are not derivable.

Definition A.2 *A semi-oblivious sanitizable signcryption scheme* $\mathsf{sOSSC} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{sig}}, \mathsf{KeyGen}_{\mathsf{san}}, \mathsf{SignEnc}, \mathsf{SanDec}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *is* immutable *if for every ppt adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{Immutability}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda) \ ,$$

where the probability is taken over the random coins of $\mathcal{A}$ *and the* Immutability *experiment which is defined as follows:*

$\mathsf{Exp}^{\mathsf{Immutability}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda)$

$(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$;
$(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(\mathsf{Msk})$;
$(pk^*_{\mathsf{san}}, \widetilde{\mathsf{m}^*\sigma^*}) \leftarrow \mathcal{A}^{\mathsf{SignEnc}(\cdot,\cdot,sk_{\mathsf{sig}},\cdot,\mathsf{pparam}),\mathsf{Proof}(sk_{\mathsf{sig}},\cdot,\cdot,\cdot)}(pk_{\mathsf{sig}}, \mathsf{pparam})$;
if $(\mathsf{Verify}(\widetilde{\mathsf{m}^*\sigma^*}, pk_{\mathsf{sig}}, pk^*_{\mathsf{san}}) = 1 \wedge$ *"*$\widetilde{\mathsf{m}^*\sigma^*}$ *is not derivable"*
$\wedge$ *"*$\widetilde{\mathsf{m}^*\sigma^*} \leftarrow \mathsf{SignEnc}(m, \mathrm{ADM}^*, sk_{\mathsf{sig}}, pk^*_{\mathsf{san}}, \mathsf{pparam})$ *was not queried for any* m *and* ADM^* *where* ADM^* *is recoverable from* σ^**"*) **then**
return 1.
else return 0.

Here, "$\widetilde{\mathsf{m}^*\sigma^*}$ *is not derivable" means that* $\mathsf{SanDec}(c_{(m,\sigma)}, \mathrm{MOD}, pk_{\mathsf{sig}}, sk^*_{\mathsf{san}}, \mathsf{pparam})$ *does not yield the signature* $\widetilde{\mathsf{m}^*\sigma^*}$ *for any* MOD *and any* $c_{(m,\sigma)}$ *under* sk_{sig} *and* pk^*_{san} *where* sk^*_{san} *is the secret key corresponding to* pk^*_{san}.

Privacy is probably the most important goal for a sanitizable scheme. If data is sanitized, we want to be sure that this data cannot be recovered in any efficient way. For a signature scheme it is formalized by requiring that given two messages m_0 and m_1, as well as a description of admissible parts ADM, and modifications MOD_0 and MOD_1, respectively, we demand that it is infeasible to distinguish which message was sanitized given (m_b, σ_b) whenever $\mathrm{MOD}_0(m_0) = \mathrm{MOD}_1(m_1)$. In the strong version, the adversary receives the sanitizer's public key as additional input yielding an "insider-version[5]" of the property. If we encrypt message-signature pairs in the first step of the sanitization process (from signer to sanitizer) we still have the same security requirements: data which was deleted in the sanitization process should remain unrecoverable from the derived message-signature pair (apart from what is derivable from the context, of course).

Definition A.3 *A semi-oblivious sanitizable signcryption scheme* $\mathsf{sOSSC} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{sig}}, \mathsf{KeyGen}_{\mathsf{san}}, \mathsf{SignEnc}, \mathsf{SanDec}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *is* strongly private *if for every ppt adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{SPrivacy}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda) \ ,$$

where the probability is taken over the random coins of $\mathcal{A}$ *and the* SPrivacy *experiment which is defined as follows:*

$\mathsf{Exp}^{\mathsf{SPrivacy}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda)$

$(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$;
$(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(\mathsf{Msk})$;
$(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(\mathsf{Msk})$;
$b \leftarrow \{0,1\}$;
$b' \leftarrow \mathcal{A}^{\mathsf{SignEnc}(\cdot,\cdot,sk_{\mathsf{sig}},\cdot,\mathsf{pparam}),\mathsf{Proof}(sk_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{LoRSSanit}(\cdot,\cdot,sk_{\mathsf{sig}},sk_{\mathsf{san}},\cdot,b,\mathsf{pparam})}(pk_{\mathsf{sig}}, sk_{\mathsf{san}}, \mathsf{pparam})$;
if $b = b'$ **then**
return 1.
else return 0.

$\mathsf{LoRSSanit}((m_0, m_1), \mathrm{ADM}, sk_{\mathsf{sig}}, sk_{\mathsf{san}}, (\mathrm{MOD}_0, \mathrm{MOD}_1), b, \mathsf{pparam})$

if $(\mathrm{ADM}(\mathrm{MOD}_0) \neq 1 \vee \mathrm{ADM}(\mathrm{MOD}_1) \neq 1 \vee \mathrm{MOD}_0(m_0) \neq \mathrm{MOD}_1(m_1))$ **then**

  return $\perp$.

else

  $c_{(m_b,\sigma_b)} \leftarrow \mathsf{SignEnc}(m_b, \mathrm{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam})$;

  $(m'_b, \sigma'_b) \leftarrow \mathsf{SanDec}(c_{(m_b,\sigma_b)}, \mathrm{MOD}_b, sk_{\mathsf{san}}, pk_{\mathsf{sig}}, \mathsf{pparam})$;

  return (m'_b, σ'_b).

For our purpose, we require *perfect* privacy. For this, we require aditionally that sanitized signatures are independently distributed from the original message.

Recall that unlinkability ensures that a sanitized message-signature pair cannot be linked to its originating document. Likewise, such a notion is desirable in the context of semi-oblivious sanitizable signcryption schemes. As for privacy, we focus on the stronger version, namely strengthened unlinkability, and the regular version can be easily derived by designating a fixed key pair for the signer and giving pk_{sig} to the adversary as additional input.

Definition A.4 *A semi-oblivious sanitizable signcryption scheme* $\mathsf{sOSSC} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{sig}}, \mathsf{KeyGen}_{\mathsf{san}}, \mathsf{SignEnc}, \mathsf{SanDec}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *is* strongly unlinkable *if for every ppt adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{SUnlinkability}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda) \ ,$$

where the probability is taken over the random coins of $\mathcal{A}$ *and the* SUnlinkability *experiment which is defined as follows:*

$\mathsf{Exp}^{\mathsf{SUnlinkability}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda)$

$(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$;

$(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(\mathsf{Msk})$;

$b \leftarrow \{0, 1\}$;

$b' \leftarrow \mathcal{A}^{\mathsf{SanDec}(\cdot,\cdot,sk_{\mathsf{san}},\cdot,\mathsf{pparam}),\mathsf{LoRSanDec}(\cdot,\cdot,sk_{\mathsf{san}},\cdot,b,\mathsf{pparam})}(pk_{\mathsf{san}}, \mathsf{pparam})$;

if $b = b'$ **then**

  return 1.

else return 0.

$\mathsf{LoRSanDec}(c_{(m_0,\sigma_0)}, c_{(m_1,\sigma_1)}, pk_{\mathsf{sig}}, sk_{\mathsf{san}}, (\mathrm{MOD}_0, \mathrm{MOD}_1), b, \mathsf{pparam})$

if $\mathrm{MOD}_0(m_0) \neq \mathrm{MOD}_1(m_1)$ **then**

  return $\perp$.

else

  $(m'_0, \sigma'_0) \leftarrow \mathsf{SanDec}(c_{(m_0,\sigma_0)}, \mathrm{MOD}_0, sk_{\mathsf{san}}, pk_{\mathsf{sig}}, \mathsf{pparam})$;

  $(m'_1, \sigma'_1) \leftarrow \mathsf{SanDec}(c_{(m_1,\sigma_1)}, \mathrm{MOD}_1, sk_{\mathsf{san}}, pk_{\mathsf{sig}}, \mathsf{pparam})$;

return (m'_b, σ'_b).

Note that LoRSanDec *will sanitize both messages to check if the description of admissible parts recovered from both signatures matches the respective modification. If this is not the case then* SanDec*, thus also* LoRSanDec*, returns an error.*

The last security property inherent to sanitizable signatures, we have not yet considered, is accountability which says that is is desirable to be able to determine which party generated the given signature. More precisely, accountability ensures that a malicious party (signer or sanitizer) cannot accuse the other party of being responsible for a message he created.

If we think of the primitive of semi-oblivious sanitizable signcryption, we might ask ourselves why we need a notion of accountability, since commonly it is easy to distinguish between an encrypted and a decrypted message-signature tuple. If all parties behaved honestly signers are only able to generate encrypted tuple, as their signing procedure is chained to an encryption process, whereas honest sanitizers can only produce decrypted pairs, since the sanitization procedure is linked to a decryption key. However, when we think of security properties, we have to assume that parties will exploit whatever possibilities they are given. For example, in our generic construction, a malicious signer, being in possession of the decryption key $\mathsf{sk}_{\mathsf{ID}}$ can always decrypt a tuple he created and accuse the sanitizer. On the other hand, a sanitizer can retrieve the encryption key EK_1 from pparam and re-encrypt the derived pair in order to blame the signer. This should convince the reader that the nature of a semi-oblivious sanitizable signcryption scheme does not alone suffice to provide accountability.

Definition A.5 *A semi-oblivious sanitizable signcryption scheme* $\mathsf{sOSSC} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{sig}}, \mathsf{KeyGen}_{\mathsf{san}}, \mathsf{SignEnc}, \mathsf{SanDec}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *is* non-interactive publicly accountable *if the* Proof *algorithm always returns* $\perp$ *and for every ppt adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{PubAccountability}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda) \ ,$$

where the probability is taken over the random coins of $\mathcal{A}$ *and the* PubAccountability *experiment which is defined as follows:*

$\mathsf{Exp}^{\mathsf{PubAccountability}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda)$

$(\mathsf{Msk}, \mathsf{pparam}) \leftarrow \mathsf{Setup}(1^\lambda)$;
$(pk_{\mathsf{sig}}, sk_{\mathsf{sig}}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(\mathsf{Msk})$;
$(pk_{\mathsf{san}}, sk_{\mathsf{san}}) \leftarrow \mathsf{KeyGen}_{\mathsf{san}}(\mathsf{Msk})$;
$(pk^*, \widetilde{\mathsf{m}^*\sigma^*}) \leftarrow \mathcal{A}^{\mathsf{SignEnc}(\cdot,\cdot,sk_{\mathsf{sig}},\cdot,\mathsf{pparam}),\mathsf{SanDec}(\cdot,\cdot,sk_{\mathsf{san}},\cdot,\mathsf{pparam})}(pk_{\mathsf{sig}}, pk_{\mathsf{san}}, \mathsf{pparam})$;
if $(\mathsf{Verify}(\widetilde{\mathsf{m}^*\sigma^*}, pk_{\mathsf{sig}}, pk^*, \mathsf{pparam}) = 1 \wedge \mathsf{Judge}(\widetilde{\mathsf{m}^*\sigma^*}, pk_{\mathsf{sig}}, pk^*, \perp, \mathsf{pparam}) = 1$
$\wedge$ *"no oracle query of the form* $\widetilde{\mathsf{m}^*\sigma^*} \leftarrow \mathsf{SignEnc}(m, \mathrm{ADM}, sk_{\mathsf{sig}}, pk^*, \mathsf{pparam})$
was made for any m *and* ADM*"*) **then**
return 1.
else if $(\mathsf{Verify}(\widetilde{\mathsf{m}^*\sigma^*}, pk^*, pk_{\mathsf{san}}, \mathsf{pparam}) = 1 \wedge \mathsf{Judge}(\widetilde{\mathsf{m}^*\sigma^*}, pk^*, pk_{\mathsf{san}}, \perp, \mathsf{pparam}) = 0$
$\wedge$ *"no oracle query of the form* $\widetilde{\mathsf{m}^*\sigma^*} \leftarrow \mathsf{SanDec}(c_{(m,\sigma)}, \mathrm{MOD}, sk_{\mathsf{san}}, pk^*, \mathsf{pparam})$
was made for any $c_{(m,\sigma)}$ *and* MOD*"*) **then**
return 1.
else return 0.

B Proof of Theorem 3.4 (Security of Our Semi-Oblivious Sanitizable Signcryption Scheme)

By definition of a semi-oblivious sanitizable signcryption, sOSSC, given in Construction 3.2, cannot fulfill transparency. The reason is simply because signatures honestly created by a signer will always be encrypted, whereas any honestly derived signature will appear in the clear. This is due to the specific structure of the primitive: in a semi-oblivious sanitizable signcryption scheme there only exists a combined signing and encryption algorithm, namely $\mathsf{SignEnc}$. No distinct signing algorithm is specified. Likewise, a sanitizer can only make use of SanDec to scrutinize a message. He cannot solely decrypt or sanitize. We would like to stress that for Construction 3.2 we do have distinct algorithms for each of the aforementioned procedures if we look at the underlying $\mathcal{S}$ and FE schemes. However, the adversary would always win the experiment as any challenge is honestly generated.

For the rest of the security properties, we can show consecutively that each is implied by one of the properties of the underlying schemes.

Claim B.1 *The semi-oblivious sanitizable signcryption scheme* sOSSC *in Construction 3.2 is (strongly) unforgeable if the underlying sanitizable signature scheme* $\mathcal{S}$ *used is (strongly) unforgeable.*

Proof. Assume that the scheme sOSSC from Construction 3.2 is not unforgeable. Then there exists an efficient adversary $\mathcal{A}$ which has a non-negligible advantage of winning the $\mathsf{SUnforgeability}$ experiment described in Definition A.1. Using $\mathcal{A}$ as a subroutine, we can construct an adversary $\mathcal{B}$ against the (strong) unforgeability of the underlying sanitizable signature scheme $\mathcal{S}$ as follows: upon input of a signer's and a sanitizer's public key pk_{sig} and pk_{san}, the adversary $\mathcal{B}$ chooses a multi-input functional encryption scheme FE, runs its $\mathsf{FE.Setup}$ algorithm on integer 4 to receive a master secret key Msk as well as encryption keys $\mathbf{EK} = (\mathsf{EK}_1, \ldots, \mathsf{EK}_4)$, generates decryption keys $\mathsf{sk}_{\mathsf{Vf}}$ and sk_{J} by calling $\mathsf{FE.KeyGen}$ with input Msk and the respective circuit $\mathsf{S.Verify}(\cdot,\cdot,\cdot)$ and $\mathsf{S.Judge}(\cdot,\cdot,\cdot,\cdot)$, which can easily be done as the description of the algorithms of $\mathcal{S}$ are public, and combines these values to derive public parameters pparam. Separately, $\mathcal{B}$ also uses Msk to derive a general decryption key $\mathsf{sk}_{\mathsf{ID}}$ for the circuit $\mathsf{ID}(\cdot)$ capable of decrypting any ciphertext. Now, the adversary $\mathcal{B}$ initializes $\mathcal{A}$ on input pk_{sig}, pk_{san}, and pparam. Queries from $\mathcal{A}$ are handled as follows:

> *SignEnc Queries.* For every query of the form $(m, \mathrm{ADM}, pk'_{\mathsf{san}})$ to $\mathcal{A}$'s $\mathsf{SignEnc}$ oracle, $\mathcal{B}$ forwards the query unchanged to his own Sign oracle and receives a message-signature pair (m, σ). Then, he retrieves encryption key EK_1 from pparam and encrypts the two values using $\mathsf{FE.Enc}$ to obtain $c_{(m,\sigma)} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_1, \sigma)$. The value $c_{(m,\sigma)}$ is returned to $\mathcal{A}$.
>
> *SanDec Queries.* For every query of the form $(c_{(m,\sigma)}, \mathrm{MOD}, pk'_{\mathsf{sig}})$ to $\mathcal{A}$'s SanDec oracle, $\mathcal{B}$ first decrypts the pair $c_{(m,\sigma)}$ using the previously generated $\mathsf{sk}_{\mathsf{ID}}$ to retrieve (m, σ), then forwards $((m, \sigma), \mathrm{MOD}, pk'_{\mathsf{sig}})$ to his Sanit oracle, and finally returns the answer (m', σ') unmodified to $\mathcal{A}$.
>
> *Proof Queries.* Every query $(\widetilde{\mathsf{m}\sigma}, \{\widetilde{\mathsf{m}_\mathsf{i}\sigma_\mathsf{i}}\}_{i\in\{1,\ldots,t\}}, pk'_{\mathsf{san}})$ to $\mathcal{A}$'s Proof oracle is answered by returning $\perp$ to $\mathcal{A}$. The adversaries Proof oracle is not needed for generating the response, since the Proof algorithm of Construction 3.2 always returns the empty string.

Finally, $\mathcal{A}$ outputs a tuple $\widetilde{\mathsf{m}^*\sigma^*}$. Now, $\mathcal{B}$ checks if $\mathsf{S.Verify}(\widetilde{\mathsf{m}^*\sigma^*}, pk_{\mathsf{sig}}, pk_{\mathsf{san}})$ returns true. If so, he outputs $\widetilde{\mathsf{m}^*\sigma^*}$. Else, he decrypts the tuple using $\mathsf{sk}_{\mathsf{ID}}$ and returns the decrypted pair.

Obviously, $\mathcal{B}$ is efficient since $\mathcal{A}$ is efficient by assumption, oracle queries can be done in constant time, and all procedures of FE are ppt. Furthermore, he simulates all oracles for $\mathcal{A}$ perfectly[7] resulting in

[7] Note that the simulation is only *perfect* if the underlying functional encryption scheme is perfectly correct. If we allowed a functional encryption scheme to have a negligible error when decrypting, we would obtain a chance of an incorrect simulation.

$\mathcal{A}$ outputting a valid forgery for sOSSC with non-negligible probability. Since the structure of the Verify algorithm of sOSSC permits two possibilities for the structure of $\widetilde{\mathsf{m}^*\sigma^*}$ to return 1, namely, either if S.Verify will accept the pair, or if the decryption under $\mathsf{sk}_{\mathsf{Vf}}$, which represents the output of S.Verify on input the decrypted version, (m^*, σ^*), returns true on the tuple, $\mathcal{A}$ can either return an encrypted tuple of a valid signature pair, or a decrypted one to win the game. In the later case, the S.Verify algorithm will directly return 1 upon input of this tuple, and $\mathcal{B}$ will thus have found a valid forgery for $\mathcal{S}$. If $\widetilde{\mathsf{m}^*\sigma^*}$ is indeed encrypted, then by decrypting it with $\mathsf{sk}_{\mathsf{ID}}$, $\mathcal{B}$ receives a message-signature pair which will evaluate S.Verify to true due to the structure of Verify.

To win, $\mathcal{A}$ did not ask for any SignEnc query which returned $(\widetilde{\mathsf{m}^*\sigma^*}, \text{ADM}^*)$. This means that also $\mathcal{B}$ will not have made a query of the form (m^*, ADM^*) to get $\sigma^* \leftarrow \mathsf{Sign}(1^\lambda, m^*, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \text{ADM}^*)$. Furthermore, $\mathcal{A}$ did not ask for any SanDec query which returned $\widetilde{\mathsf{m}^*\sigma^*}$ which in turn implies that also $\mathcal{B}$ did not query his Sanit oracle a query yielding this pair. Hence, all conditions are fulfilled and $\mathcal{B}$ wins $\mathsf{Exp}^{\mathsf{SUnforgeability}}_{\mathcal{A},\mathcal{S}}(\lambda)$ if and only if $\mathcal{A}$ wins $\mathsf{Exp}^{\mathsf{SUnforgeability}}_{\mathcal{A},\mathsf{sOSSC}}(\lambda)$, i.e., the probability that $\mathcal{B}$ wins is equivalent to the probability of $\mathcal{A}$ winning and, therefore, not negligible either. This is a contradiction to $\mathcal{S}$ being (strongly) unforgeable, meaning that the assumption that sOSSC is not (strongly) unforgeable must have been false. The claim equally holds if we only consider regular unforgeability.

Claim B.2 *The semi-oblivious sanitizable signcryption* sOSSC *in Construction 3.2 is immutable if the underlying sanitizable signature scheme $\mathcal{S}$ used is immutable.*
Proof. The proof is analogous to the above treating unforgeability. The only difference is that we do not have a SanDec or Sanit oracle, and therefore, do not need to simulate these.

Claim B.3 *The semi-oblivious sanitizable signcryption* sOSSC *in Construction 3.2 is (strongly) private if the underlying sanitizable signature scheme $\mathcal{S}$ used is (strongly) private.*
Proof. Assume that the scheme sOSSC from Construction 3.2 is not (strongly) private. Then there exists an efficient adversary $\mathcal{A}$ which has a non-negligible advantage of winning the SPrivacy experiment described in Definition A.3. Using $\mathcal{A}$ as a subroutine, we can construct an adversary $\mathcal{B}$ against the (strong) privacy property of the underlying sanitizable signature scheme $\mathcal{S}$ as follows: upon input of a signer's public key pk_{sig}, and a sanitizer's key pair $(pk_{\mathsf{san}}, sk_{\mathsf{san}})$, the adversary $\mathcal{B}$ chooses a multi-input functional encryption scheme FE, runs its FE.Setup algorithm on integer 4 to receive a master secret key Msk as well as encryption keys $\mathbf{EK} = (\mathsf{EK}_1, \ldots, \mathsf{EK}_4)$, generates decryption keys $\mathsf{sk}_{\mathsf{Vf}}$, sk_{J} by calling FE.KeyGen with input Msk and the respective circuit $\mathsf{S.Verify}(\cdot,\cdot,\cdot)$ and $\mathsf{S.Judge}(\cdot,\cdot,\cdot,\cdot)$, and combines these values to derive public parameters pparam. Furthermore, he calls FE.KeyGen with input Msk and the sanitization circuit in which he first hard-codes the sanitizer's secret key to derive a sanitizer's secret decryption key, i.e., $\mathsf{sk}_{\mathsf{Sanit}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{Msk}, \mathsf{S.Sanit}(\cdot,\cdot,sk_{\mathsf{san}},\cdot))$. Separately, $\mathcal{B}$ also uses Msk to derive a general decryption key $\mathsf{sk}_{\mathsf{ID}}$ for the circuit $\mathsf{ID}(\cdot)$ capable of decrypting any ciphertext. Now, adversary $\mathcal{B}$ initializes $\mathcal{A}$ on input pk_{sig}, $(pk_{\mathsf{san}}, \mathsf{sk}_{\mathsf{Sanit}})$, and pparam. Queries from $\mathcal{A}$ are handled as follows:

> *SignEnc Queries.* For every query of the form $(m, \text{ADM}, pk'_{\mathsf{san}})$ to $\mathcal{A}$'s SignEnc oracle, $\mathcal{B}$ forwards the query unchanged to his own Sign oracle and receives a message-signature pair (m, σ). Then, he retrieves encryption key EK_1 from pparam and encrypts the value using FE.Enc to obtain $c_{(m,\sigma)} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_1, (m, \sigma))$. The value $c_{(m,\sigma)}$ is returned to $\mathcal{A}$.
>
> *Proof Queries.* Every query $(\widetilde{\mathsf{m}\sigma}, \{\widetilde{\mathsf{m_i}\sigma_\mathsf{i}}\}_{i\in\{1,\ldots,t\}}, pk'_{\mathsf{san}})$ to $\mathcal{A}$'s Proof oracle is answered by returning $\perp$ to $\mathcal{A}$. The adversaries Proof oracle is not needed.

Hence, also the resulting success probability of $\mathcal{B}$ would be slightly less then if the scheme were perfectly correct. However, since this loss is only negligible we would still achieve the same results.

Challenge Queries. Every query $((m_0, m_1), \text{ADM}, (\text{MOD}_0, \text{MOD}_1))$ to $\mathcal{A}$'s $\mathsf{LoRSSanit}$ oracle is directly forwarded to $\mathcal{B}$'s $\mathsf{LoRSSanit}$ oracle. The answer (m'_b, σ'_b) is forwarded unmodified to $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a bit b' representing his decision which message was sanitized. The adversary $\mathcal{B}$ returns the same value b'. Obviously, $\mathcal{B}$ is efficient since $\mathcal{A}$ is efficient by assumption, oracle queries can be done in constant time, and all procedures of FE are ppt. Furthermore, he perfectly simulates all oracles for $\mathcal{A}$ resulting in $\mathcal{A}$ outputting the correct decision, i.e., if $b' = b$, with non-negligible probability. We see that $\mathcal{B}$ wins the $\mathsf{SPrivacy}$ experiment for semi-oblivious sanitizable signcryption schemes if and only if $\mathcal{A}$ wins the sanitizable signature scheme version of the (strong) privacy experiment, since $\mathcal{B}$ will not have asked any unauthorized queries unless $\mathcal{A}$ has asked such. Hence, the success probability of $\mathcal{B}$ is equal to the success probability of $\mathcal{A}$ and, therefore, not negligible either which contradicts the assumption that $\mathcal{S}$ is (strongly) private. Thus, the assumption must have been wrong and sOSSC is (strongly) private. The claim holds equally if we only consider regular privacy.

Claim B.4 *The semi-oblivious sanitizable signcryption* sOSSC *in Construction 3.2 is (strongly) unlinkable if the underlying sanitizable signature scheme* $\mathcal{S}$ *used is (strongly) unlinkable.*
Proof. Assume that the scheme sOSSC from Construction 3.2 is not (strongly) unlinkable. Then there exists an efficient adversary $\mathcal{A}$ which has a non-negligible advantage of winning the $\mathsf{SUnlinkability}$ experiment described in Definition A.4. Using $\mathcal{A}$ as a subroutine, we can construct an adversary $\mathcal{B}$ against the strengthened unlinkability property of the underlying sanitizable signature scheme $\mathcal{S}$ as follows: upon input of a sanitizer's public key pk_{san}, the adversary $\mathcal{B}$ chooses a multi-input functional encryption scheme FE, runs its $\mathsf{FE.Setup}$ algorithm on integer 4 to receive a master secret key Msk as well as encryption keys $\mathbf{EK} = (\mathsf{EK}_1, \ldots, \mathsf{EK}_4)$, generates decryption keys $\mathsf{sk}_{\mathsf{Vf}}$ and sk_{J} by calling $\mathsf{FE.KeyGen}$ with input Msk and the respective circuit $\mathsf{S.Verify}(\cdot,\cdot,\cdot)$ and $\mathsf{S.Judge}(\cdot,\cdot,\cdot,\cdot)$, and combines these values to derive public parameters pparam. Separately, $\mathcal{B}$ also uses Msk to derive a general decryption key $\mathsf{sk}_{\mathsf{ID}}$ for the circuit $\mathsf{ID}(\cdot)$ capable of decrypting any ciphertext. Now, the adversary $\mathcal{B}$ initializes $\mathcal{A}$ on input pk_{san} and pparam. Queries from $\mathcal{A}$ are handled as follows:

SignEnc Queries. For every query of the form $(m, \text{ADM}, pk'_{\mathsf{san}})$ to $\mathcal{A}$'s $\mathsf{SignEnc}$ oracle, $\mathcal{B}$ forwards the query unchanged to his own Sign oracle and receives a message-signature pair (m, σ). Then, he retrieves encryption key EK_1 from pparam and encrypts the value using $\mathsf{FE.Enc}$ to obtain $c_{(m,\sigma)} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_1, (m, \sigma))$. The value $c_{(m,\sigma)}$ is returned to $\mathcal{A}$.

SanDec Queries. For every query of the form $(c_{(m,\sigma)}, \text{MOD}, pk_{\mathsf{sig}})$ to $\mathcal{A}$'s SanDec oracle, $\mathcal{B}$ first uses his decryption key $\mathsf{sk}_{\mathsf{ID}}$ to decrypt the values and retrieve (m, σ). He forwards this tuple with MOD and pk_{sig} to his Sanit oracle. The oracle returns a derived message-signature pair (m', σ') which is passed on to $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a bit b' representing his decision which message was sanitized which $\mathcal{B}$ outputs, too. Obviously, $\mathcal{B}$ is efficient since $\mathcal{A}$ is efficient by assumption, oracle queries can be done in constant time, and all procedures of FE are ppt. Furthermore, he perfectly simulates all oracles for $\mathcal{A}$ resulting in $\mathcal{A}$ outputting the correct decision, i.e., if $b' = b$, with non-negligible probability.We see that $\mathcal{B}$ wins the $\mathsf{SUnlinkability}$ experiment for semi-oblivious sanitizable signcryption schemes if and only if $\mathcal{A}$ wins the sanitizable signature scheme version of the strengthened unlinkability experiment, since $\mathcal{B}$ will not have asked any unauthorized queries unless $\mathcal{A}$ has asked such. Hence, the success probability of $\mathcal{B}$ is equal to the success probability of $\mathcal{A}$, and therefore not negligible either, which contradicts the assumption that $\mathcal{S}$ is (strongly) unlinkable. Thus, the assumption must have been wrong and sOSSC is (strongly) unlinkable. The claim holds equally if we only consider regular unlinkability.

Claim B.5 *The semi-oblivious sanitizable signcryption scheme* sOSSC *in Construction 3.2 is non-interactive publicly accountable if the underlying sanitizable signature scheme $\mathcal{S}$ used is non-interactive publicly accountable.*

Proof. Assume that the scheme sOSSC from Construction 3.2 is not non-interactive publicly accountable. Then there exists an efficient adversary $\mathcal{A}$ which has a non-negligible advantage of winning the PubAccountability experiment described in Definition A.5. Using $\mathcal{A}$ as a subroutine, we can construct an adversary $\mathcal{B}$ against the non-interactive public accountability property of the underlying sanitizable signature scheme $\mathcal{S}$ as follows: upon input of a signer's and sanitizer's public key pk_{sig} and pk_{san}, the adversary $\mathcal{B}$ chooses a multi-input functional encryption scheme FE, runs its FE.Setup algorithm on integer 4 to receive a master secret key Msk as well as encryption keys $\mathbf{EK} = (\mathsf{EK}_1, \ldots, \mathsf{EK}_4)$, generates decryption keys $\mathsf{sk}_{\mathsf{Vf}}$ and sk_{J}, by calling FE.KeyGen with input Msk and the respective circuit $\mathsf{S.Verify}(\cdot,\cdot,\cdot)$ and $\mathsf{S.Judge}(\cdot,\cdot,\cdot,\cdot)$, and combines these values to derive public parameters pparam. Separately, $\mathcal{B}$ also uses Msk to derive a general decryption key $\mathsf{sk}_{\mathsf{ID}}$ for the circuit $\mathsf{ID}(\cdot)$ capable of decrypting any ciphertext. Now, the adversary $\mathcal{B}$ initializes $\mathcal{A}$ on input pk_{sig}, pk_{san}, and pparam. Queries from $\mathcal{A}$ are handled as follows:

> *SignEnc Queries.* For every query of the form $(m, \text{ADM}, pk'_{\mathsf{san}})$ to $\mathcal{A}$'s SignEnc oracle, $\mathcal{B}$ forwards the query unchanged to his own Sign oracle and receives a message-signature pair (m, σ). Then, he retrieves encryption key EK_1 from pparam and encrypts the value using FE.Enc to obtain $c_{(m,\sigma)} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_1, (m, \sigma))$. The value $c_{(m,\sigma)}$ is returned to $\mathcal{A}$.
>
> *SanDec Queries.* For every query of the form $(c_{(m,\sigma)}, \text{MOD}, pk_{\mathsf{sig}})$ to $\mathcal{A}$'s SanDec oracle, $\mathcal{B}$ first uses his decryption key $\mathsf{sk}_{\mathsf{ID}}$ to decrypt the values and retrieve (m, σ). He forwards this tuple with MOD and pk_{sig} to his Sanit oracle. The oracle returns a derived message-signature pair (m'_b, σ'_b) which is passed on to $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a tuple $(pk^*, \widetilde{\mathsf{m}^*\sigma^*})$. Now, $\mathcal{B}$ checks if either $\mathsf{S.Verify}(\widetilde{\mathsf{m}^*\sigma^*}, pk_{\mathsf{sig}}, pk^*)$ or $\mathsf{S.Verify}(\widetilde{\mathsf{m}^*\sigma^*}, pk^*, pk_{\mathsf{san}})$ returns true. If so, he outputs $(pk^*, \widetilde{\mathsf{m}^*\sigma^*})$. Else, he decrypts the tuple using $\mathsf{sk}_{\mathsf{ID}}$ and returns the decrypted pair together with the target public key pk^*. Obviously, $\mathcal{B}$ is efficient since $\mathcal{A}$ is efficient by assumption, oracle queries can be done in constant time, and all procedures of FE are ppt. Furthermore, he perfectly simulates all oracles for $\mathcal{A}$ resulting in $\mathcal{A}$ outputting a tuple which wins the BlockPubAccountability experiment with non negligible probability.

Now, we will take a look at the success probability of $\mathcal{B}$. Observe that, whenever $\mathcal{A}$ wins the game, either he returns a tuple which blames the signer for creating a signature which was actually created by the sanitizer, i.e., $\mathsf{Verify}(\widetilde{\mathsf{m}^*\sigma^*}, pk_{\mathsf{sig}}, pk^*, \mathsf{pparam}) = 1 \wedge \mathsf{Judge}(\widetilde{\mathsf{m}^*\sigma^*}, pk_{\mathsf{sig}}, pk^*, \perp, \mathsf{pparam}) = 1 \wedge$ "no oracle query of the form $\widetilde{\mathsf{m}^*\sigma^*} \leftarrow \mathsf{SignEnc}(m, \text{ADM}, sk_{\mathsf{sig}}, pk_{\mathsf{san}}\, \mathsf{pparam})$ was made for any m and ADM") holds, or he returns a tuple which blames the sanitizer even though the signer is accountable, i.e., we have $\mathsf{Verify}(\widetilde{\mathsf{m}^*\sigma^*}, pk^*, pk_{\mathsf{san}}, \mathsf{pparam}) = 1 \wedge \mathsf{Judge}(\widetilde{\mathsf{m}^*\sigma^*}, pk^*, pk_{\mathsf{san}}, \perp, \mathsf{pparam}) = 0 \wedge$ "no oracle query of the form $\widetilde{\mathsf{m}^*\sigma^*} \leftarrow \mathsf{SanDec}(c_{(m,\sigma)}, \text{MOD}, sk_{\mathsf{san}}, pk^*, \mathsf{pparam})$ was made for any $c_{(m,\sigma)}$, and MOD"). We see that these constraints translate directly to the case of $\mathcal{S}$, since if the pair $\widetilde{\mathsf{m}^*\sigma^*}$ is not encrypted, then S.Verify will accept it under pk^* and the corresponding signer's or sanitizer's public key. $\mathcal{B}$ performs both checks on S.Verify since he might not know whether or not the pair is encrypted and to which party the target public key belongs[8]. In order for $\mathcal{B}$ to win his own experiment, he needs to output a valid signature such that S.Judge will accuse the wrong party. Whenever $\mathcal{A}$ wins his game, S.Judge will do so since the decision of Judge basically forwards the decision of S.Judge by definition, and finally, no illegitimate queries will have been posed since the restriction on forbidden queries is the same for both games when regarding respected oracles. Thus, $\mathcal{B}$ always wins, whenever $\mathcal{A}$ wins and, hence, has the same non-negligible success

[8] If the adversary can determine if the tuple is encrypted or not, he could omit the last checks and either return it directly or decipher it (using $\mathsf{sk}_{\mathsf{ID}}$) and return the tuple.

probability, which contradicts the assumption that $\mathcal{S}$ is non-interactive publicly accountable, meaning, the assumption must have been wrong and sOSSC is non-interactive publicly accountable.

Claim B.6 *The semi-oblivious sanitizable signcryption scheme* sOSSC *in Construction 3.2 is* IND-CCA$_{\mathsf{san}}$*-secure if the underlying multi-input functional encryption scheme* FE *used is* miIND*-secure and the underlying sanitizable scheme $\mathcal{S}$ is perfectly private.*

Proof. Assume that the scheme sOSSC from Construction 3.2 is not IND-CCA$_{\mathsf{san}}$-secure with respect to Definition 3.3. Then there exists an efficient adversary $\mathcal{A}$ which has a non-negligible advantage of winning the IND-CCA$_{\mathsf{san}}$ experiment. Using $\mathcal{A}$ as a subroutine, we can construct an adversary $\mathcal{B}$ against the miIND security of the underlying multi-input functional encryption scheme FE as follows: upon input of security parameter 1^λ, the adversary $\mathcal{B}$ first outputs the set $I = \{1, \ldots, 4\}$ to receive a set of decryption keys $\mathbf{EK} = (\mathsf{EK}_1, \ldots, \mathsf{EK}_4)$. He instantiates a sanitizable signature scheme $\mathcal{S}$ and runs the key generation algorithms to generate $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}})$ and $(pk_{\mathsf{san}}, sk_{\mathsf{san}})$. Then, he uses his FE.KeyGen oracle to create decryption keys $\mathsf{sk}_{\mathsf{Vf}}, \mathsf{sk}_{\mathsf{J}}$, and using sk_{san} hard-coded into the S.Sanit circuit to get $\mathsf{sk}_{\mathsf{Sanit}}$. Finally, $\mathcal{B}$ derives public parameters pparam and initializes $\mathcal{A}$ by inputting a signer's public key pk_{sig}, a sanitizer's key pair $(pk_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{Sanit}})$, as well as pparam. All emerging queries are handled in the following way.

> *SignEnc Queries.* For every query of the form $(m, \text{ADM}, pk_{\mathsf{san}})$ to $\mathcal{A}$'s SignEnc oracle, $\mathcal{B}$ uses S.Sign to retrieve $(m, \sigma) \leftarrow \mathsf{S.Sign}(m, sk_{\mathsf{sig}}, pk_{\mathsf{san}}, \text{ADM})$, then uses encryption key EK_1 to encrypt message and signature, respectively, and returns $c_{(m,\sigma)}$ to $\mathcal{A}$.
>
> *Challenge Queries.* For every query of the form $((m_0, m_1), \text{ADM})$ to $\mathcal{A}$'s LoREncSign oracle, $\mathcal{B}$ first signs both messages using S.Sign with sk_{sig} and retrieves (m_0, σ_0) and (m_1, σ_1). He sends $((m_0, \sigma_0), (m_1, \sigma_1))$ to his LoREnc oracle to receive $c_{(m_b, \sigma_b)}$ and forwards the tuple to $\mathcal{A}$.

Finally, $\mathcal{A}$ will terminate and output a decision b' which is equally output by $\mathcal{B}$.

We see that $\mathcal{B}$ is efficient since $\mathcal{A}$ is efficient, oracle queries can be done in constant time, and both sampling a sanitizable signature scheme and evaluating its algorithms can be conducted in polynomial time. Furthermore, being in possession of the signer's secret key as well as all relevant encryption keys, $\mathcal{B}$ can perfectly simulate all oracle queries for $\mathcal{A}$.

At last, note that for $\mathcal{A}$ to win the IND-CCA$_{\mathsf{san}}$ game, he is restricted in the queries he is allowed to pose to the LoREncSign oracle. More precisely, any query must fulfill that there is no "splitting" modification, meaning that any possible modification which is admitted needs to yield the same message m' for the two inputs. Given this, it follows from the perfect privacy of the sanitizable scheme that also the signature part σ' to m' contained in reply of the oracle is equally distributed (or, if the scheme has been derandomized, identical). It follows that in $\mathcal{B}$'s experiment the key $\mathsf{sk}_{\mathsf{Sanit}}$ does not split the inputs to the LoREnc oracle, where the second to fourth input is taken from a singleton. This is also true for the verification and judge keys, $\mathsf{sk}_{\mathsf{Vf}}$ and sk_{J}, since these keys cannot split such pairs because any pair is a valid signature, and created by the signer.

Therefore, $\mathcal{B}$ win the miIND game if and only if $\mathcal{A}$ wins in the IND-CCA$_{\mathsf{san}}$ experiment. Thus, their success probabilities are equal and so not negligible which is a contradiction to FE being miIND which concludes our proof.