Unlinkability of Sanitizable Signatures

Christina Brzuska Marc Fischlin Anja Lehmann Dominique Schröder

Darmstadt University of Technology, Germany
www.minicrypt.de

Abstract. Sanitizable signatures allow a designated party, called the sanitizer, to modify parts of signed data such that the immutable parts can still be verified with respect to the original signer. Ateniese et al. (ESORICS 2005) discuss five security properties for such signature schemes: unforgeability, immutability, privacy, transparency and accountability. These notions have been formalized in a recent work by Brzuska et al. (PKC 2009), discussing also the relationships among the security notions. In addition, they prove a modification of the scheme of Ateniese et al. to be secure according to these notions.

Here we discuss that a sixth property of sanitizable signature schemes may be desirable: unlinkability. Basically, this property prevents that one can link sanitized message-signature pairs of the same document, thus allowing to deduce combined information about the original document. We show that this notion implies privacy, the inability to recover the original data of sanitized parts, but is not implied by any of the other five notions. We also discuss a scheme based on group signatures meeting all six security properties.

1 Introduction

For a regular signature scheme any modification of the message makes the signature for the modified message invalid. In some applications, though, it may be preferable to support message modifications such that one can still verify the authenticity of the immutable message part, and that only authorized parties can make such changes. Signature schemes having this property are called *sanitizable*, as introduced by Ateniese et al. [ACdMT05]. Related concepts have been discussed concurrently in [SBZ01, MSI^{+}03, JMSW02].

Ateniese et al. [ACdMT05] discuss the applicability of sanitizable signatures to anonymization of medical data, replacing commercials in authenticated media streams or updates of reliable routing information. They identified five desirable security properties for sanitizable signature schemes. Informally, these are:

Unforgeability. Says that no one except for the honest signer and sanitizer can create valid signatures.

Immutability. Demands that even a malicious sanitizer cannot change message parts which have not been marked as modifiable by the signer.

Privacy. Prevents an outsider to recover the original data of sanitized message parts.

Transparency. Covers the indistinguishability of signatures created by the signer or the sanitizer.

Accountability. Refers to the inability of a malicious signer or sanitizer to deny authorship.

Brzuska et al. [BFF$^+$09] define these five properties with game-based approaches formally and relate them, showing that accountability implies unforgeability and transparency implies privacy; all other properties are independent. They also prove a modification of the scheme by Ateniese et al. [ACdMT05] to be secure according to these five properties.

Unlinkability. Here we discuss that an additional property may be useful in some settings. We call this property *unlinkability* and motivate it by the following example (see also Figure 1): Assume that we have signed medical records and at some point we anonymize the data by redacting the personal information of the patients like names, addresses etc. At some other time, say for revenues reasons, we remove the actual medical treatments and leave only the personal information. Then one should not be able to link these data through the (sanitized) signatures and therefore reconstruct the full records. However, previous schemes like the one by Brzuska et al. [BFF$^+$09] and, for example, the ones in [KL06, CLM08, CJ10] in fact allow such attacks. They are usually based on chameleon hashes which remain unchanged for the sanitization step and thus allow to identify two sanitized signatures derived from the same signature through the hash value. Other constructions like the one in [MSI$^+$03] even come with an explicit document identifier, allowing to link sanitized messages easily.

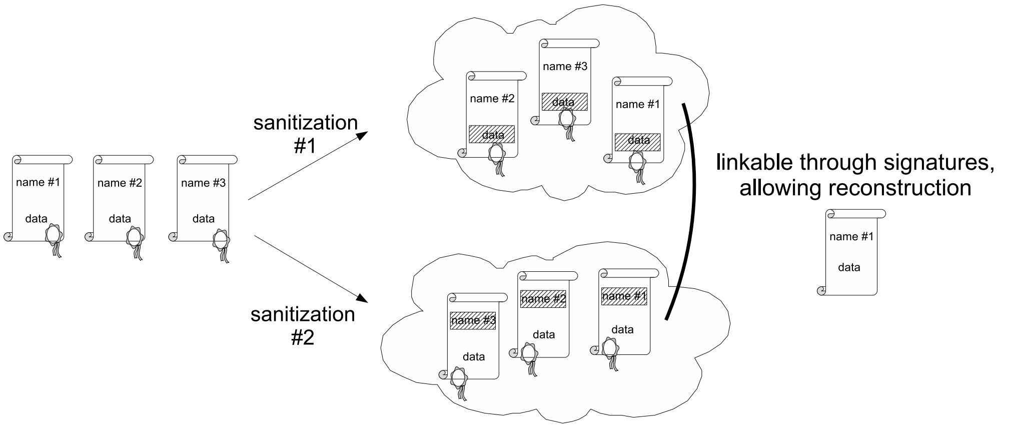

Figure 1: Linkability problem

We hence introduce a formal definition of unlinkability and relate it to the previously given notions. It turns out that unlinkability is not implied by any of the other properties, but vice versa implies privacy. The reason is that privacy prevents an adversary of recovering the original data for sanitized parts, and violation of this property also enables the adversary to reconstruct and to link messages easily.

Construction. We then present a construction of a sanitizable signature scheme obeying all six properties, including unlinkability. The idea is fundamentally different from previous approaches which usually rely on chameleon hashes. In our case the signer first signs the fixed parts with a regular signature scheme. For the modifiable parts the signer and the sanitizer use a group signature scheme [CvH91], i.e., a signature scheme which allows to sign anonymously on behalf of the group but such that a group manager can revoke the identity of the user that has signed [BMW03]. In our case the group only consists of the signer and sanitizer, and the signer also incarnates the group manager. If the sanitizer later changes (some of) the modifiable message parts it can create a new group signature and replace the signer's group signature.

The anonymity of the group signature scheme in our context guarantees transparency (the indistinguishability of signatures originating from the signer and the sanitizer). The possibility to identify a group member by the group manager (i.e., the signer in our case) supports sanitizer-accountability, i.e., the ability to provide a proof that the sanitizer has created the signature. Signer-accountability is provided by the non-frameability of the group signature scheme which prevents a malicious group manager (i.e., the signer) from falsely accusing the sanitizer to be the source of a signature. Immutability follows from the unforgeability of the regular signature scheme for the fixed parts, and unlinkability from the fact that the sanitizer signs the entire message from scratch (the signature for fixed message parts remains unchanged).

We remark that the actual construction needs a careful implementation of the idea above to make the derived sanitizable signature scheme satisfy all desired security properties. This is in particular true since proposed group signature schemes in the literature like [BMW03, BSZ05, KY05, DP06, Gro06, Gro07] come with varying security features and set-up assumptions. In this version we thus present a simple but not necessarily the most practical approach to turn our idea into a secure sanitizable scheme, e.g., following the definitions in [BFF$^+$09] we do not rely on the fact that public keys of the signer or sanitizer are registered, although this is most likely in practice.

Our solution shows that, in general, sanitizable signatures can be built from group signatures, thereby providing a new application for the latter primitive. This relation also immediately gives a feasibility result for sanitizable signatures: Since the work by Bellare et al. [BMW03] about group signatures proves that one can derive them from IND-CCA secure encryption, non-interactive zero-knowledge proofs and digital signatures, all known to exist given trapdoor permutations, it follows that one can also build secure sanitizable signatures from trapdoor permutations.

Organization. In Section 2 we introduce the notion of sanitizable signatures and the security properties given in [ACdMT05, BFF$^+$09]. In Section 3 we discuss the notion of unlinkability and its relationship to the other security properties. In Section 4 we present our construction of a secure sanitizable scheme based on group signatures. We discuss variations thereof in Section 5.

2 Preliminaries

In this section we revisit the notion of sanitizable signatures and the previously given security properties.

2.1 Sanitizable Signatures

In a sanitizable signature scheme both the signer and the sanitizer hold a key pair $(sk_{\mathrm{sig}}, pk_{\mathrm{sig}})$, $(sk_{\mathrm{san}}, pk_{\mathrm{san}})$ such that the signer can sign messages with its secret key sk_{sig} and "attach" a description of the admissible modifications $\textsc{adm}$ which are allowed to the sanitizer pk_{san}. The sanitizer can then later change such a message according to some modification $\textsc{mod}$ and update the signature using his secret key sk_{san}. In order to settle disputes about the origin of a message-signature pair the algorithm Proof enables the signer to produce a proof π from previously signed messages that a signature has been created by the sanitizer. This proof can then be verified with the help of the Judge algorithm (but which only needs to decide about the origin in case of a valid message-signature pair in question; for invalid pairs such decisions are in general impossible).

To model admissible modifications we assume that $\textsc{adm}$ and $\textsc{mod}$ are (descriptions of) efficient deterministic algorithms such that $\textsc{mod}$ maps any message m to the modified message $m' = \textsc{mod}(m)$, and $\textsc{adm}(\textsc{mod}) \in \{0,1\}$ indicates if the modification is admissible and matches $\textsc{adm}$, in which case $\textsc{adm}(\textsc{mod}) = 1$. For example, for messages $m = m[1] \dots m[k]$ divided into blocks $m[i]$ of equal bit length t we can let $\textsc{adm}$ contain t and the indices of the modifiable blocks, and $\textsc{mod}$ then essentially consists of pairs $(j, m'[j])$ defining the new value for the j-th block.

For ease of notation we let $\textsc{fix}_{\textsc{adm}}$ be an efficient deterministic algorithm which is uniquely determined by $\textsc{adm}$ and which maps m to the immutable message part $\textsc{fix}_{\textsc{adm}}(m)$, e.g., for block-divided messages $\textsc{fix}_{\textsc{adm}}(m)$ is the concatenation of all blocks not appearing in $\textsc{adm}$. To exclude trivial examples we demand that admissible modifications leave the fixed part of a message unchanged, i.e., $\textsc{fix}_{\textsc{adm}}(m) = \textsc{fix}_{\textsc{adm}}(\textsc{mod}(m))$ for all $m \in \{0,1\}^*$ and all $\textsc{mod}$ with $\textsc{adm}(\textsc{mod}) = 1$. Analogously, to avoid choices like $\textsc{fix}_{\textsc{adm}}$ having empty output, we also require that the fixed part must be "maximal" given $\textsc{adm}$, i.e., $\textsc{fix}_{\textsc{adm}}(m') \neq \textsc{fix}_{\textsc{adm}}(m)$ for $m' \notin \{\textsc{mod}(m) \mid \textsc{mod} \text{ with } \textsc{adm}(\textsc{mod}) = 1\}$.

Jumping ahead, we note that for our construction based on group signatures we make another assumption on $\textsc{adm}$. This property, denoted modification-decidability, allows to decide efficiently for given messages m, m^* and $\textsc{adm}$ whether m^* is an admissible modification of m with respect to $\textsc{adm}$ or not. This property is for example satisfied for the block-based approach. However, for our definitions of the security properties and their relationships we do not impose any restriction at this point.

The following definition is taken from [BFF$^+$09]:

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

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

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

Signing. *The Sign algorithm takes as input a message $m \in \{0,1\}^*$, the secret key sk_{sig} of the signer, the public key pk_{san} of the sanitizer, as well as a description $\textsc{adm}$ of the admissibly modifiable message parts. It outputs a signature (or $\perp$, indicating an error):*

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

We assume that $\textsc{adm}$ is recoverable from any signature $\sigma \neq \perp$.

Sanitizing. *Algorithm Sanit takes a message $m \in \{0,1\}^*$, a signature σ, the public key pk_{sig} of the signer and the secret key sk_{san} of the sanitizer. It modifies the message m according to the modification instruction $\textsc{mod}$ and determines a new signature σ' for the modified message $m' = \textsc{mod}(m)$. Then Sanit outputs m' and σ' (or possibly $\perp$ in case of an error).*

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

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

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

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

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

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

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

For a sanitizable signature scheme the usual correctness properties should hold, saying that genuinely signed or sanitized messages are accepted and that a genuinely created proof by the signer leads the judge to decide in favor of the signer. For a formal approach to correctness see [BFF$^+$09].

2.2 Security of Sanitizable Signatures

Here we recall the security notions for sanitizable signatures given by Brzuska et al. [BFF$^+$09]. We note that, there, they show that signer and sanitizer accountability together imply unforgeability, and that transparency implies privacy. Hence, in principle it suffices to show immutability, accountability and transparency.

Unforgeability. Unforgeability demands that no outsider should be able to forge signatures under the keys of the honest signer and sanitizer, i.e., no adversary should be able to compute a tuple (m^*, σ^*) such that $\mathsf{Verify}(m^*, \sigma^*, pk_{\mathrm{sig}}, pk_{\mathrm{san}}) = \mathtt{true}$ without having the secret keys $sk_{\mathrm{sig}}, sk_{\mathrm{san}}$. This must hold even if one can see additional signatures for other input data, including the message-signature pairs and the public keys. We also give the adversary access to a Proof oracle, because proofs could potentially leak information about the secret signing key. Yet, except for this secret key the adversary fully determines the other input data, including the message-signature pairs and the public keys. This allows to capture for example scenarios where several sanitizers are assigned to the same signer.

Definition 2.2 (Unforgeability) *A sanitizable signature scheme* SanSig *is* unforgeable *if for any efficient algorithm* $\mathcal{A}$ *the probability that the following experiment returns* 1 *is negligible (as a function of* n*):*

```
***Experiment*** $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathsf{SanSig}}(n)$
    $(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^n)$
    $(pk_{san}, sk_{san}) \leftarrow \mathsf{KGen}_{san}(1^n)$
    $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{sig}, \cdot, \cdot), \mathsf{Sanit}(\cdot, \cdot, \cdot, \cdot, sk_{san}), \mathsf{Proof}(sk_{sig}, \ldots, \cdot)}(pk_{sig}, pk_{san})$
```

```
        letting $(m_i, \textsc{adm}_i, pk_{san,i})$ and $\sigma_i$ for $i = 1, 2, \ldots, q$
        denote the queries and answers to and from oracle Sign,
        and $(m_j, \textsc{mod}_j, \sigma_j, pk_{sig,j})$ and $(m'_j, \sigma'_j)$ for $j = q+1, \ldots, r$
        denote the queries and answers to and from oracle Sanit.
    return 1 iff
        $\mathsf{Verify}(m^*, \sigma^*, pk_{sig}, pk_{san}) = \texttt{true}$ and
        for all $i = 1, 2, \ldots, q$ we have $(pk_{san}, m^*) \neq (pk_{san,i}, m_i)$ and
        for all $j = q+1, \ldots, r$ we have $(pk_{sig}, m^*) \neq (pk_{sig,j}, m'_j)$.
```

Immutability. This property demands informally that a malicious sanitizer cannot change inadmissible blocks. In the attack model below the malicious sanitizer $\mathcal{A}$ interacts with the signer to receive signatures σ_i for messages m_i, descriptions $\textsc{adm}_i$ and keys $pk_{\text{san},i}$ of its choice, before eventually outputting a valid pair $(pk^*_{\text{san}}, m^*, \sigma^*)$ such that message m^* is not a "legitimate" transformation of one of the m_i's under the same key $pk^*_{\text{san}} = pk_{\text{san},i}$. The latter is formalized by requiring that for each query $pk^*_{\text{san}} \neq pk_{\text{san},i}$ or $m^* \notin \{\textsc{mod}(m_i) \mid \textsc{mod} \text{ with } \textsc{adm}_i(\textsc{mod}) = 1\}$ for the value $\textsc{adm}_i$ in σ_i, e.g., that for block-divided messages m^* and m_i differ in at least one inadmissible block. As the adversary can query the signer for several sanitizer keys $pk_{\text{san,i}}$, the security definition also covers the case where the signer interacts with several sanitizers simultaneously.

Definition 2.3 (Immutability) *A sanitizable signature scheme* SanSig *is* immutable *if for any efficient algorithm $\mathcal{A}$ the probability that the following experiment* $\mathsf{Immutability}^{\mathsf{SanSig}}_{\mathcal{A}}(n)$ *returns* 1 *is negligible (as a function of n).*

```
***Experiment*** $\mathsf{Immutability}^{\mathsf{SanSig}}_{\mathcal{A}}(n)$
    $(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KGen}_{sig}(1^n)$
    $(pk^*_{san}, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, sk_{sig}, \cdot, \cdot), \mathsf{Proof}(sk_{sig}, \ldots, \cdot)}(pk_{sig})$
        letting $(m_i, \textsc{adm}_i, pk_{san,i})$ and $\sigma_i$ for $i = 1, 2, \ldots, q$
        denote the queries and answers to and from oracle Sign.
    return 1 if
        $\mathsf{Verify}(m^*, \sigma^*, pk_{sig}, pk^*_{san}) = \texttt{true}$ and
        for all $i = 1, 2, \ldots, q$ we have
            $pk^*_{san} \neq pk_{san,i}$ or
            $m^* \notin \{\textsc{mod}(m_i) \mid \textsc{mod}$ with $\textsc{adm}_i(\textsc{mod}) = 1\}$
```

Accountability. Accountability says the origin of a (sanitized) signature should be undeniable. There are the following two types of accountability: *sanitizer-accountability* says that, if a message has not been signed by the signer, then even a malicious sanitizer should not be able to make the judge accuse the signer. *Signer-accountability* says that, if a message and its signature have not been sanitized, then even a malicious signer should not be able to make the judge accuse the sanitizer.

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

Definition 2.4 (Sanitizer-Accountability) *A sanitizable signature scheme* SanSig *is said to be* sanitizer-accountable *if for any efficient algorithm* $\mathcal{A}_{\mathsf{Sanit}}$ *the probability that the experiment* $\mathsf{San\text{-}Accountability}_{\mathcal{A}_{\mathsf{Sanit}}}^{\mathsf{SanSig}}(n)$ *below returns* 1 *is negligible (as a function of* n*).*

```
Experiment San-Accountability$_{\mathcal{A}_{Sanit}}^{\mathsf{SanSig}}(n)$
    $(pk_{sig}, sk_{sig}) \leftarrow KGen_{sig}(1^n)$
    $(pk^*_{san}, m^*, \sigma^*) \leftarrow \mathcal{A}_{Sanit}^{Sign(\cdot, sk_{sig}, \cdot, \cdot), Proof(sk_{sig}, \ldots, \cdot)}(pk_{sig})$
        letting $(m_i, \textsc{adm}_i, pk_{san,i})$ and $\sigma_i$ for $i = 1, 2, \ldots, q$
        denote the queries and answers to and from oracle Sign
    $\pi \leftarrow Proof(sk_{sig}, m^*, \sigma^*, (m_1, \sigma_1), \ldots, (m_q, \sigma_q), pk^*_{san})$
    return 1 iff
        $(pk^*_{san}, m^*) \neq (pk_{san,i}, m_i)$ for all $i = 1, 2, \ldots, q$, and
        $Verify(m^*, \sigma^*, pk_{sig}, pk^*_{san}) = \texttt{true}$, and
        $Judge(m^*, \sigma^*, pk_{sig}, pk^*_{san}, \pi) = \texttt{Sig}$
```

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

Definition 2.5 (Signer-Accountability) *A sanitizable signature scheme* SanSig *is* signer-accountable *if for any efficient* $\mathcal{A}_{\mathsf{Sign}}$ *the probability that the experiment* $\mathsf{Sig\text{-}Accountability}_{\mathcal{A}_{\mathsf{Sign}}}^{\mathsf{SanSig}}(n)$ *below returns* 1 *is negligible (as a function of* n*):*

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

Privacy. Privacy roughly means that it should be infeasible to recover information about the sanitized parts of the message. As information leakage through the modified message itself can never be prevented, we only refer to information which is available through the sanitized signature.

We present here the indistinguishability notion from [BFF+09] where an adversary can choose pairs $(m_0, \textsc{mod}_0)$, $(m_1, \textsc{mod}_1)$ of messages and modifications together with a description $\textsc{adm}$ and has access to a “left-or-right” box.[1] This oracle either returns a sanitized signature for the left tuple ($b = 0$) or for the right tuple ($b = 1$). The task of the attacker is to predict the random bit b significantly better than by guessing. Here we need the additional constraint that for each call to the

[1] Brzuska et al. [BFF+09] also discuss a simulation-based approach which is equivalent to the indistinguishability notion.

left-or-right box the resulting modified messages are identical for both tuples and the modifications both match $\textsc{adm}$, else the task would be trivial. We write $(m_0, \textsc{mod}_0, \textsc{adm}) \equiv (m_1, \textsc{mod}_1, \textsc{adm})$ for this.

Definition 2.6 (Privacy) *A sanitizable signature scheme* SanSig *is* private *if for any efficient algorithm $\mathcal{A}$ the probability that the following experiment returns 1 is negligibly close to $\frac{1}{2}$:*

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

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

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

Adversary $\mathcal{A}$ eventually produces an output a, the guess for b. A sanitizable signature is now said to be *transparent* if for all efficient algorithms $\mathcal{A}$ the probability for a right guess $a = b$ in the above game is negligibly close to $\frac{1}{2}$. Below we also define a relaxed version called *proof-restricted transparency* and discuss the idea after the definition.

Definition 2.7 ((Proof-Restricted) Transparency) *A sanitizable signature scheme* SanSig *is* (proof-restricted) transparent *if for any efficient algorithm $\mathcal{A}$ the probability that the following experiment* $\mathsf{Transparency}_{\mathcal{A}}^{\mathsf{SanSig}}(n)$ *returns 1 is negligibly close to $\frac{1}{2}$.*

```
***Experiment*** $\mathsf{Transparency}_{\mathcal{A}}^{\mathsf{SanSig}}(n)$
    $(pk_{sig}, sk_{sig}) \leftarrow \mathit{KGen}_{sig}(1^n)$
    $(pk_{san}, sk_{san}) \leftarrow \mathit{KGen}_{san}(1^n)$
    $b \leftarrow \{0,1\}$
    $a \leftarrow \mathcal{A}^{\mathit{Sign}(\cdot,sk_{sig},\cdot,\cdot),\mathit{Sanit}(\cdot,\cdot,\cdot,\cdot,sk_{san}),\mathit{Proof}(sk_{sig},\ldots,\cdot),\mathit{Sanit/Sign}(\cdot,\cdot,\cdot,sk_{sig},sk_{san},pk_{sig},pk_{san},b)}(pk_{sig}, pk_{san})$
        where oracle $\mathit{Sanit/Sign}$ for input $m_k, \textsc{mod}_k, \textsc{adm}_k$
        first computes $\sigma_k \leftarrow \mathsf{Sign}(m_k, sk_{sig}, pk_{san}, \text{ADM}_k)$,
        then computes $(m'_k, \sigma'_k) \leftarrow \mathsf{Sanit}(m_k, \text{MOD}_k, \sigma_k, pk_{sig}, sk_{san})$,
        then, if $b = 1$, replaces $\sigma'_k$ by $\sigma'_k \leftarrow \mathsf{Sign}(m'_k, sk_{sig}, pk_{san}, \text{ADM}_k)$,
        and finally returns $(m'_k, \sigma'_k)$.
    return 1 iff
        $a = b$
        (and, in the proof-restricted case, $\mathcal{A}$ has not queried
        any $m'_k$ output by $\mathsf{Sanit}/\mathsf{Sign}$ to $\mathsf{Proof}$)
```

The original definition of Brzuska et al. [BFF$^+$09] does not consider the proof-restricted case. Without this restriction, though, achieving transparency at first seems to be impossible because the adversary can then always submit the replies of the Sanit/Sign oracle to the Proof oracle and thereby recover the secret bit b. However, in their construction the Proof algorithm searches in the list of previously signed messages and only gives a useful answer if it finds a match, enabling transparency without this restriction. Yet, any solution (like ours here) where the Proof algorithm is "history-free" can only achieve the proof-restricted version. Note that Proof algorithms forgoing the set of previously signed messages are preferable from an efficiency point of view, of course.

As for the implications among the security notions [BFF$^+$09] we note that proof-restricted transparency only implies a proof-restricted form of privacy, where the answer messages of the LoRSanit oracle cannot be submitted to the Proof oracle either. However, since we show in the next section that unlinkability implies full privacy and our construction achieves unlinkability, our scheme is also private in the non-restricted sense. We note that all the separation results in [BFF$^+$09] remain valid for proof-restricted transparency.

3 Unlinkability

In this section we define unlinkability formally and discuss its relationship to the other security notions.

3.1 Definition

As explained in the introduction, unlinkability refers to the impossibility to use the signatures to identify sanitized message-signature pairs originating from the same source. Technically, we use an indistinguishability-based approach to define this property, saying that, given a signature for a sanitized message of two possible sources, the adversary cannot predict the actual original message better than by guessing. This should even hold if the adversary herself provides the two source message-signature pairs and modifications of which one is sanitized. The stipulation here is that the two modifications yield the same sanitized message. Else, if for example the sanitized messages still contain some unique but distinct entry, then predicting the source is easy, of course. This, however, is beyond the scope of signature schemes: the scheme should only prevent that *signatures* can be used to link data.

Formally, we use a game-based approach to define unlinkability, similar to the other security notions in [BFF$^+$09]. The adversary is given access to a signing oracle and a sanitizer oracle (and a proof oracle since this step depends on the signer's secret key and may leak valuable information). The adversary is also allowed to query a left-or-right oracle LoRSanit which is initialized with a secret random bit b. In each of the multiple queries to $\mathsf{LoRSanit}$ the adversary provides a pair of tuples, each consisting of a message, a modification and a *valid* signature, such that the recoverable description of admissible modifications is identical in both cases (since we assume that ADM is recoverable from a signature providing distinct descriptions ADM would allow a trivial attack; so would the case that only one signature is valid). Depending on the bit b, the adversary gets the sanitized message-signature pair of either the left or right input pair. The adversary should eventually predict the bit b significantly better than with the guessing probability of $\frac{1}{2}$.

Definition 3.1 (Unlinkability) *A sanitizable signature scheme* SanSig *is* unlinkable *if for any efficient algorithm $\mathcal{A}$ the probability that the following experiment $\mathsf{Unlinkability}_{\mathcal{A}}^{\mathsf{SanSig}}(n)$ returns 1 is negligibly close to $\frac{1}{2}$.*

```
***Experiment*** $\mathsf{Unlinkability}_{\mathcal{A}}^{\mathsf{SanSig}}(n)$
    $(pk_{sig}, sk_{sig}) \leftarrow \mathit{KGen}_{sig}(1^n)$
    $(pk_{san}, sk_{san}) \leftarrow \mathit{KGen}_{san}(1^n)$
    $b \leftarrow \{0,1\}$
    $a \leftarrow \mathcal{A}^{\mathit{Sign}(\cdot, sk_{sig}, \cdot, \cdot), \mathit{Sanit}(\cdot,\cdot,\cdot,\cdot, sk_{san}), \mathit{Proof}(sk_{sig}, \cdots, \cdot), \mathit{LoRSanit}(\cdot,\cdot, sk_{sig}, sk_{san}, b)}(pk_{sig}, pk_{san})$
        where oracle $\mathit{LoRSanit}(\cdot, \cdot, sk_{sig}, sk_{san}, b)$, on input
        $(m_{j,0}, \text{MOD}_{j,0}, \sigma_{j,0})$,$(m_{j,1}, \text{MOD}_{j,1}, \sigma_{j,1})$ with recoverable $\text{ADM}_{j,0} = \text{ADM}_{j,1}$
        $\mathit{Verify}(m_{j0}, \sigma_{j,0}, pk_{sig}, pk_{san}) = \texttt{true}$, $\mathit{Verify}(m_{j1}, \sigma_{j,1}, pk_{sig}, pk_{san}) = \texttt{true}$,
        returns $(m'_j, \sigma'_j) \leftarrow \mathit{Sanit}(m_{j,b}, \text{MOD}_{j,b}, \sigma_{j,b}, pk_{sig}, sk_{san})$,
        and where $(m_{j,0}, \text{MOD}_{j,0}, \text{ADM}_j) \equiv (m_{j,1}, \text{MOD}_{j,1}, \text{ADM}_j)$,
        i.e., $m_{j,0}$ and $m_{j,1}$ are mapped to the same modified message.
    return 1 if $a = b$.
```

A pictorial description is given in Figure 2. We note that the definition above is for example robust concerning several sanitization steps in the $\mathsf{LoRSanit}$ oracle. That is, we could allow the adversary in the experiment above to submit arbitrarily long "modification chains" $\text{MOD}_{j,0}^{1}, \ldots, \text{MOD}_{j,0}^{m}$ and $\text{MOD}_{j,1}^{1}, \ldots, \text{MOD}_{j,1}^{m}$ such that the two source documents are gradually sanitized with a match in the resulting documents. Still, predicting b remains hard, as such chains can potentially be simulated by calling the sanitizer oracle for the first $m-1$ modifications manually, before entering the final sanitization step into the $\mathsf{LoRSanit}$ oracle. Jumping ahead, we remark that this property also allows to show that unlinkability actually implies stronger notions of privacy than defined in [BFF$^+$09] (see the next section).

Recall the example of medical records which are sanitized twice, one time basically removing the personal information and the other time removing the medical data. Our notion of unlinkability can then be used to show that such sanitized message-signature pairs do not allow to reconstruct the full data better than by guessing. Assume for simplicity that we only have two records with entries $(\texttt{name\#0}, \texttt{data\#0})$ and $(\texttt{name\#1}, \texttt{data\#1})$. Then we create all four possible combinations $(\texttt{name\#}a, \texttt{data\#}b)$ for $a, b \in \{0,1\}$ and ask for signatures for them (with both parts being admissibly changeable). For each $a \in \{0,1\}$ we then insert the pairs $(\texttt{name\#}a, \texttt{data\#0})$ and $(\texttt{name\#}a, \texttt{data\#1})$ twice into the $\mathsf{LoRSanit}$ oracle, one time cutting off the name-part, the other time removing the data-part. Altogether we make thus four calls to the $\mathsf{LoRSanit}$ oracle, and we hand those four replies to the adversary. Our unlinkability definition says that one cannot distinguish the two cases (left or right sanitization) better than by guessing, thus also disallowing to tell which data belong to whose name.

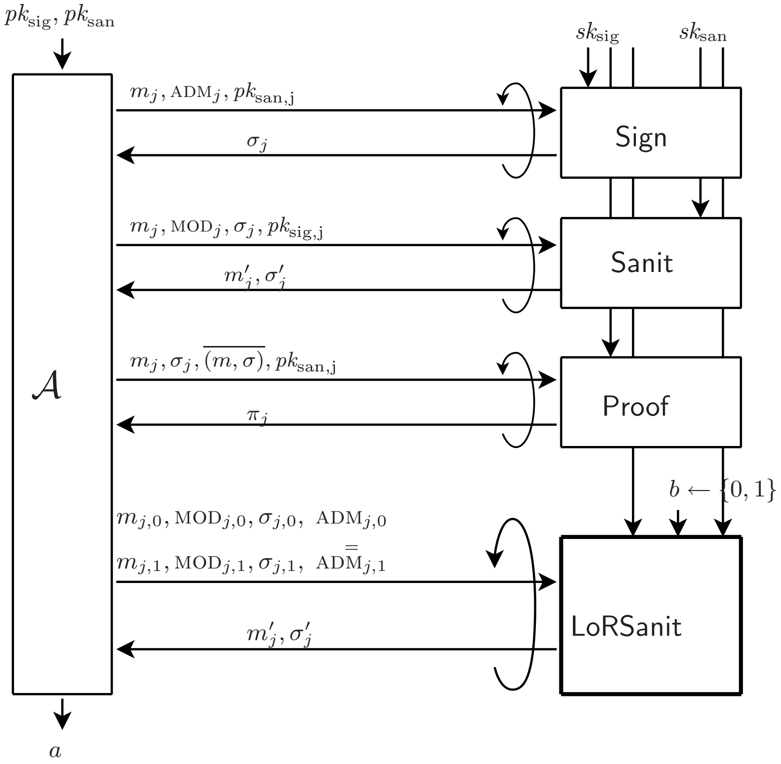
$pk_{\text{sig}}, pk_{\text{san}}$
sk_{sig}
sk_{san}
$m_j, \text{ADM}_j, pk_{\text{san,j}}$
σ_j
Sign
$m_j, \text{MOD}_j, \sigma_j, pk_{\text{sig,j}}$
m'_j, σ'_j
Sanit
$\mathcal{A}$
$m_j, \sigma_j, \overline{(m, \sigma)}, pk_{\text{san,j}}$
π_j
Proof
$b \leftarrow \{0, 1\}$
$m_{j,0}, \text{MOD}_{j,0}, \sigma_{j,0}, \text{ADM}_{j,0}$
$=$
$m_{j,1}, \text{MOD}_{j,1}, \sigma_{j,1}, \text{ADM}_{j,1}$
m'_j, σ'_j
LoRSanit
a

Figure 2: Unlinkability. $\mathcal{A}$ wins if it outputs $a = b$.

Our definition above is for unlinkability with respect to message-signature pairs sanitized by the *same* sanitizer. One can easily extend the above definition by demanding that the adversary can also determine different sanitizers for the left and for the right input data. But then both sanitizers must have been declared to have the permission to sanitize, otherwise one could easily determine the secret bit of the LoRSanit by picking an invalid sanitizer for one of the input tuples.

3.2 Relationships of the Security Notions

We first show that unlinkability does not follow from any of the other security requirements. Then we prove that unlinkability implies privacy, and finally discuss that unlinkability does not imply any of the other properties.

Proposition 3.2 (Independence of Unlinkability) *Assume that there exists a sanitizable signature scheme (obeying one or more of the properties unforgeability, immutability, privacy, (proof-restricted) transparency, signer-accountability and sanitizer-accountability). Then there exists a sanitizable signature scheme which is not unlinkable but preserves the other security properties.*

Proof. We show this proposition by modifying a sanitizable signature scheme possibly fulfilling the other security requirements in a way that unlinkability no longer holds, but all the other requirements remain unaffected. The idea is to augment signatures by unique identifiers such that sanitied documents can be easily linked. Let $\mathsf{SanSig} = (\mathsf{KGen}_{\text{sig}}, \mathsf{KGen}_{\text{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$

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

- $\mathsf{KGen}_{\mathrm{sig}}$ and $\mathsf{KGen}_{\mathrm{san}}$ remain unchanged.
- Sign' works as Sign, except that it appends a random value id $\leftarrow \{0,1\}^n$ to the signature, $\sigma' = \sigma||\mathrm{id}$.
- Sanit' works as Sanit, except that it first chops off the final n bits id of the input signature, then runs Sanit and finally appends id to the resulting signature again.
- all other algorithms merely chop off the final n bits of any input signature and then work as their ancestors (but using the shortened signatures).

It is clear that unforgeability, immutability and accountability are not affected by the modification of the signature value above. As for privacy note that the signatures for the left or right message contain the same identifier, and (proof-restricted) transparency follows because in the final signatures (which is either first signed and then sanitized, or signed from scratch) the identifier is an independent random element. Unlinkability does not hold because a sanitized documents inherits the (quasi) unique identifier from the original signature, allowing to determine the origin easily.

More formally, the adversary against unlinkability calls the signing oracle twice about the same message m, the input key pk_{san} and the same ADM, obtaining two signatures $\sigma_0||\mathrm{id}_0$ and $\sigma_1||\mathrm{id}_0$ with distinct $\mathrm{id}_0 \neq \mathrm{id}_1$ (the case $\mathrm{id}_0 = \mathrm{id}_1$ only happens with probability 2^{-n}). It picks some admissible modification MOD (say, the one which leaves the message unchanged) and submits $(m, \mathrm{MOD}, \sigma_0||\mathrm{id}_0, m, \mathrm{MOD}, \sigma_1||\mathrm{id}_1)$ to the $\mathsf{LoRSanit}$ oracle to obtain a sanitized message with signature $\sigma'_b||\mathrm{id}_b$. The adversary outputs $a = 0$ iff id_0 matches the identifier in this signature, such that a matches b with overwhelming probability. □

Proposition 3.3 (Unlinkability Implies Privacy) *Any unlinkable sanitizable signature scheme is also private.*

Proof. Assume that such a signature was not private. Then consider an adversary $\mathcal{A}_{\mathrm{priv}}$ being able to win in the privacy game with probability significantly better than $\frac{1}{2}$. We show that we we can build an adversary $\mathcal{A}_{\mathrm{unlink}}$ against unlinkability, winning with the same probability as $\mathcal{A}_{\mathrm{priv}}$.

Note that the only difference between the two attacks lies in the possibility of $\mathcal{A}_{\mathrm{unlink}}$ to provide the signatures to the $\mathsf{LoRSanit}$ oracle, whereas for $\mathcal{A}_{\mathrm{priv}}$ this signature is created inside the $\mathsf{LoRSanit}$-box from scratch (using the supplied ADM_j). This can be easily simulated by $\mathcal{A}_{\mathrm{unlink}}$ by calling the signer oracle sequentially about the two messages pairs (with identical ADM_j) to first derive a signature for each message, and then to submit these two signatures as part of the $\mathsf{LoRSanit}$-query. It follows that the success probability of $\mathcal{A}_{\mathrm{unlink}}$ is identical to the one of $\mathcal{A}_{\mathrm{priv}}$. □

The proof above shows that unlinkability actually implies something stronger than the privacy notion. Namely, instead of calling the $\mathsf{LoRSanit}$ oracle in the privacy experiments about message pairs and single modifications, one may now call this oracle about modification chains (with the stipulation that the final messages are identical) and possibly receive the intermediate signatures as additional output. Since one can easily simulate such upstream modifications in the unlinkability experiment with the help of the signing and sanitizing oracle, privacy in this case follows again from our unlinkability notion.

With the next proposition we show that unlinkability does not imply any of the other security properties (assuming that we start with a secure sanitizable signature scheme like the one we construct in the next section):

Proposition 3.4 (Independence of Other Properties) *Assume that there exists a sanitizable signature scheme which is unforgeable, immutable, private, (proof-restricted) transparent, signer-accountable, sanitizer-accountable and unlinkable. Then for any of the properties immutability, transparency, unlinkability, signer-accountability and sanitizer-accountability, there exists a sanitizable signature scheme obeying all properties except for the one in question.*

Proof. The fact that unlinkability does not follow from the other properties has already been shown in Proposition 3.2. For the other properties we remark that the counterexamples in [BFF$^+$09] which seperate immutability, transparency, signer-accountability and sanitizer-accountability from the other properties also preserve unlinkability in each case (and also hold for proof-restricted transparency). □

4 Constructions based on Group Signatures

In this section we present our unlinkable sanitizable signature scheme (which also satisfies the other security properties). As explained in the introduction, the idea is to use a group signature scheme for the group consisting of the signer and the sanitizer, such that the signer signs the immutable message part with a regular signature scheme and the full message with the group signature scheme. The sanitizer can then update the full message and only sign this second component. The signer also takes over the role of the group manager in order to provide accountability.

4.1 Group Signatures

Group signatures, introduced by Chaum and van Heyst [CvH91], allow a set of users to sign on behalf of the group such that outsiders cannot distinguish between different signers (anonymity) but such that a group manager can trace the signer's identity (traceability). We follow the formal framework of Bellare et al. [BMW03] but add the non-frameability requirement [BSZ05] that even the group manager cannot sign on behalf of the users. Recall that this is necessary for the accountability in our sanitizable signature scheme, where the signer acts as the group manager and should not be able to make the judge falsely accuse the sanitizer.

Definition 4.1 (Group Signature) *A group signature scheme GS consists of six efficient algorithms $\mathsf{GS} = (\mathsf{GKGen}, \mathsf{UKGen}, \mathsf{GSig}, \mathsf{GVf}, \mathsf{Open}, \mathsf{GJudge})$, where $\mathsf{GKGen}, \mathsf{UKGen}$ are only invoked during the setup.*

Setup phase. *There are two key generation algorithms. Let k be the number of group members. Each user i, $1 \leq i \leq k$ runs UKGen to generate a key pair. Formally, UKGen gets the security parameter 1^n as input and returns a key pair:*

$$(sk_{user,i}, pk_{user,i}) \leftarrow \mathsf{UKGen}(1^n).$$

We write $\boldsymbol{gsk}_{user}$ for the tuple $(sk_{user,1}, \ldots, sk_{user,k})$. The group manager's key generation algorithm now computes the group manager's secret key $gmsk$ as well as the public verification *key gpk for group signatures, and certificates* $\mathsf{\textbf{cert}} = (\mathsf{cert}_1, \ldots, \mathsf{cert}_k)$ *for each of the user's public key* $\boldsymbol{gpk}_{user} = (pk_{user,1}, \ldots, pk_{user,k})$*:*

$$(gmsk, gpk, \mathsf{\textbf{cert}}) \leftarrow \mathsf{GKGen}(1^n, \boldsymbol{gpk}_{user})$$

Each certificate is passed to the corresponding user.

Signing. *To create a signature for a message* $m \in \{0,1\}^*$ *the users's private key as well as the matching certificate and the group's public key gpk are required:*

$$\sigma \leftarrow \mathsf{GSig}(sk_{user,i}, cert_i, gpk, m).$$

Verification. *A signature* σ *for a message* m *is verifiable with the group's public key gpk:*

$$d \leftarrow \mathsf{GVf}(gpk, m, \sigma)$$

with $d \in \{\mathtt{true}, \mathtt{false}\}$*.*

Opening messages. *In order to settle disputes about a signature's origin, the group manager can produce a proof for the judge, usually suggesting which group member* i *signed the message:*

$$(i, \pi) \leftarrow \mathsf{Open}(gmsk, m, \sigma, \boldsymbol{gpk}_{user}, gpk)$$

where possibly $(i, \pi) = \perp$*.*

Judge. *Depending on the proof presented by the group manager, the judge decides about the signature's origin:*

$$j \leftarrow \mathsf{GJudge}(m, \sigma, i, \pi, gpk, \boldsymbol{gpk}_{user})$$

with $j \in \{\mathtt{true}, \mathtt{false}, \perp\}$*.*

The usual correctness properties should hold, i.e., genuinely generated signatures shall be accepted be the verification algorithm, and group manager proofs for honestly signed messages should be confirmed by the judge. More formally,

Signing Correctness. *For any security parameter* $n \in \mathbb{N}$*, any* $k \in \mathbb{N}$*, any tuple of user key pairs* $(sk_{user,1}, pk_{user,1}), \ldots, (sk_{userk}, pk_{userk}) \leftarrow \mathsf{UKGen}(1^n)$ *issued by* UKGen *and any output* $(gmsk, gpk, \mathsf{\textbf{cert}}) \leftarrow \mathsf{GKGen}(1^n, \boldsymbol{gpk}_{user})$*, we have that for every message* $m \in \{0,1\}^*$ *and any* $\sigma \leftarrow \mathsf{GSig}(sk_{user,i}, \mathsf{cert}_i, m)$ *for* $i \in \{1, 2, \ldots, k\}$ *it holds* $\mathsf{GVf}(gpk, m, \sigma) = \mathtt{true}$*.*

Proof Correctness. *For any security parameter* $n \in \mathbb{N}$*, any* $k \in \mathbb{N}$*, any tuple of user key pairs* $(sk_{user,1}, pk_{user,1}), \ldots, (sk_{userk}, pk_{userk}) \leftarrow \mathsf{UKGen}(1^n)$ *issued by* UKGen *and any output* $(gmsk, gpk, \mathsf{\textbf{cert}}) \leftarrow \mathsf{GKGen}(1^n, \boldsymbol{gpk}_{user})$*, it is guaranteed that for every message* $m \in \{0,1\}^*$*, any* $\sigma \leftarrow \mathsf{GSig}(sk_{user,i}, \mathsf{cert}_i, m)$ *for* $i \in \{1, 2, \ldots, k\}$ *and every proof* $(i, \pi) \leftarrow \mathsf{Open}(gmsk, m, \sigma, gpk)$ *we have* $\mathsf{GJudge}(m, \sigma, i, \pi, gpk, \boldsymbol{gpk}_{user}) = \mathtt{true}$*.*

Anonymity of group signatures means that it should be infeasible for outsiders to determine who generated the signature. This group signature property later ensures the transparency property of our sanitizable signature scheme. To simplify we give the definition only with respect to two users (the signer and the sanitizer in our case) and without corruptions, since we only care about transparency against outsiders. We, nonetheless, still give the adversary the user's secret keys and certificates but could alternatively only grant it oracle access to corresponding signing oracles:

Definition 4.2 (Anonymity) *A group signature scheme $\mathsf{GS} = (\mathsf{GKGen}, \mathsf{UKGen}, \mathsf{GSig}, \mathsf{GVf}, \mathsf{Open}, \mathsf{GJudge})$ satisfies* (full) anonymity *if for any efficient algorithm $\mathcal{B}$ the probability that the following experiment returns* 1 *is negligibly close to $\frac{1}{2}$ (as a function of n):*

```
$\textbf{\textit{Experiment}}\ \mathsf{Exp}^{anon}_{\mathsf{GS},\mathcal{B}}(n)$
    $(sk_{user,0}, pk_{user,0}), (sk_{user,1}, pk_{user,1}) \leftarrow \mathsf{UKGen}(1^n)$
    $(gmsk, gpk, \mathbf{cert}) \leftarrow \mathsf{GKGen}(1^n, \boldsymbol{gpk}_{user})$
    $b \leftarrow \{0,1\}$
    $a \leftarrow \mathcal{B}^{\mathsf{Open}(gmsk,\cdot,\cdot,\boldsymbol{gpk}_{user},gpk),\mathsf{LoRSign}(\boldsymbol{gsk}_{user},\mathbf{cert},gpk,\cdot,b)}(gpk, \boldsymbol{gsk}_{user}, \mathbf{cert})$
        where oracle $\mathsf{LoRSign}$ on input $m_i$ returns
        $\sigma_i \leftarrow \mathsf{GSig}(sk_{userb}, \mathsf{cert}_b, gpk, m_i)$
        and $\mathcal{B}$ never queries $\mathsf{Open}$ about any $m_i$
        having submitted in a communication with $\mathsf{LoRSign}$ before.
    return 1 iff $a = b$
```

Note that Bellare et al. [BMW03] define anonymity with respect to a single challenge signature only, whereas in our definition above we allow the adversary access to a left-or-right oracle which she can query multiple times. The single-query case implies the multiple-queries case by a standard hybrid argument. Also, we prohibit the adversary from submitting any *message* to Open if it has been used to get a signature via $\mathsf{LoRSign}$, whereas the definition in [BMW03] only demands that message-*signature* pairs (m_i, σ_i) appearing in a communication with $\mathsf{LoRSign}$ may not be submitted to Open. This relaxation possibly enlarges the pool of candidates and suffices for our setting.

Traceability of a group signature means that even malicious group members can be traced by the group manager. This property guarantees (sanitizer-)accountability for our sanitizable scheme. We again tailor the definition towards our needs and give the definition for two users only of which one (the sanitizer) is controlled by the adversary:

Definition 4.3 (Traceability) *A group signature scheme $\mathsf{GS} = (\mathsf{GKGen}, \mathsf{UKGen}, \mathsf{GSig}, \mathsf{GVf}, \mathsf{Open}, \mathsf{GJudge})$ satisfies* traceability *if for any efficient adversary $\mathcal{B}$ the probability that the following experiment returns* 1 *is negligible:*

```
$\textbf{\textit{Experiment}}\ \mathsf{Exp}^{trace}_{\mathsf{GS},\mathcal{B}}(n)$
    $(sk_{user,0}, pk_{user,0}) \leftarrow \mathsf{UKGen}(1^n)$
    $(st, pk^*_{user,1}) \leftarrow \mathcal{B}(\texttt{choose}, 1^n, pk_{user,0})$
    $(gmsk, gpk, \mathsf{cert}_0, \mathsf{cert}_1) \leftarrow \mathsf{GKGen}(1^n, pk_{user,0}, pk^*_{user1})$
    $(m^*, \sigma^*) \leftarrow \mathcal{B}^{\mathsf{GSig}(sk_{user,0},\mathsf{cert}_0,\cdot,gpk),\mathsf{Open}(gmsk,\cdot,\cdot,gpk)}(\texttt{guess}, st, gpk, \mathbf{cert})$
    return 0 if $\mathsf{GVf}(gpk, m^*, \sigma^*) = \texttt{false}$
    return 1 if $(i,\pi) = \perp$ for $(i,\pi) \leftarrow \mathsf{Open}(gmsk, m^*, \sigma^*, gpk)$
    return 1 if
        $i = 0$ and
        $\mathsf{GJudge}(m^*, \sigma^*, 0, \pi, gpk) = \texttt{true}$ and
        $\mathcal{B}$ has never queried the $\mathsf{GSig}$-oracle about $m^*$ in the $\texttt{guess}$ stage.
```

Finally, non-frameability of a group signature scheme provides signer-accountability: even a malicious signer collaborating with the group manager cannot make the sanitizer falsly accused as the source of a signature. In the definition below we let the adversary ask for signatures under the sanitizer's secret key *even for chosen group data.* This is necessary since the signer (and thus the group manager) is under control of the adversary and the sanitizer may be assigned to several signers.

Definition 4.4 (Non-Frameability) *A group signature scheme* $\mathsf{GS} = (\mathsf{GKGen}, \mathsf{UKGen}, \mathsf{GSig}, \mathsf{GVf}, \mathsf{Open}, \mathsf{GJudge})$ *is* non-frameable *if for any efficient adversary* $\mathcal{B}$ *the probability that the following experiment returns* 1 *is negligible:*

```
$\textbf{\textit{Experiment}}\ \mathsf{Exp}^{nonfr}_{\mathsf{GS},\mathcal{B}}(n)$
    $(sk_{user,1}, pk_{user,1}) \leftarrow \mathsf{UKGen}(1^n)$
    $(m^*, \sigma^*, \pi^*, gpk^*, \mathsf{cert}^*_1) \leftarrow \mathcal{B}^{\mathsf{GSig}(sk_{user,1},\cdot,\cdot,\cdot)}(1^n, pk_{user,1})$
    return 1 if
        $\mathsf{GVf}(gpk^*, m^*, \sigma^*) = \texttt{true}$ and
        $\mathsf{GJudge}(m^*, \sigma^*, 1, \pi^*, gpk^*) = \texttt{true}$ and
        $\mathcal{B}$ has not queried the $\mathsf{GSig}$-oracle about $m^*$.
```

Definition 4.5 (Secure Group Signature) *We call a group signature scheme* secure *if it is anonymous, traceable and non-frameable.*

As for instantiations we remark that the (generic) construction by Bellare et al. [BMW03] satisfies our adapted definitions. As for more efficient group signature schemes, we can implement our sanitizable signature scheme with other group signature schemes like [KY05, DP06, Gro06, Gro07]. Yet, these group signature schemes need additional set-up assumptions like a trusted party generating common parameters or interactive registration of users. Our sanitizable signature scheme then inherits these characteristics (recall that, in practice, registration of signer and sanitizer keys is for example necessary to provide meaningful accountability).

4.2 Construction

In this section we show that the new security requirement of unlinkability can be achieved in combination with the five established security properties formalized in [BFF$^+$09]. Recall that we sign the entire message, including the modifiable parts, with the group signature scheme, and —in order to prevent inadmissible changes— the signer also signs the fixed part with a regular scheme. This requires some care because if we take an arbitrary signature scheme then the signature itself may act as a unique identifier, even for messages with identical fixed parts. Thereby, unlinkability would be violated.

The solution is to use a secure deterministic signature scheme for the fixed part (such that the signature is identical for messages with the same fixed part). Alternatively, one can deploy a rerandomizable signature scheme such that the sanitizer can rerandomize the signature, excising the link to the input signature. Below we use the "deterministic solution" for simplicity, and since every secure signature scheme can be easily turned into a deterministic one via pseudorandom functions [Gol87].

Signature schemes and their security are defined in Appendix A where we in particular define *strong unforgeability* of such schemes. We need this unforgeability notion (saying that one cannot even find a new signature for a previously signed message) to provide unlinkability. Examples of signature schemes achieving this strong notion are [BR96, Cor00, BLS04, BB04, BSW06]. Moreover, it is possible to obtain a strongly unforgeable signature scheme out of any unforgeable signature scheme applying the transformation of Bellare and Shoup [BS07]. Applying the transformation of [Gol87] one can then make such schemes also deterministic.

Recall that the idea behind our scheme is that for each signature the signer uses a group manager key, creates a certified user key to sign the modifiable parts, and certifies the sanitizer's public key as a group member to support modifications. But since our definition of sanitizable signatures demands statefree solutions, the signer formally cannot store the group manager key for this sanitizer and would need to create a new one for each call. We bypass this as follows: we let the signer for each signing request, including a public key of the sanitizer pk_{san}, create the group manager's keys etc. via the corresponding group signature algorithms, but provide the randomness for these algorithms by applying a pseudorandom function to pk_{san} (see Appendix A for a definition of pseudorandom functions). By this, we end up with (almost) independent keys for different sanitizers, but use consistent parameters for each sanitizer. For the same reason we also include the group membership certificate of the sanitizer in the signature, although it would be given directly to the sanitizer instead. As a side effect, since the group manager's public key is tied to the sanitizer in question, we also rely on group signatures with static joins only.

Construction 4.6 (Sanitizable Signature Scheme) *Let $\mathcal{S} = (\mathsf{SKGen}, \mathsf{SSign}, \mathsf{SVf})$ be a (regular) signature scheme, let $\mathsf{GS} = (\mathsf{GKGen}, \mathsf{UKGen}, \mathsf{GSig}, \mathsf{GVf}, \mathsf{Open}, \mathsf{GJudge})$ be a group signature scheme. Let $\mathcal{PRF} = (\mathsf{KGen}_{prf}, \mathsf{PRF})$ be pseudorandom function. Define the sanitizable signature scheme* $\mathsf{SanSig} = (\mathsf{KGen}_{sig}, \mathsf{KGen}_{san}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *as follows:*

Key Generation. *First, algorithm KGen_{sig} gets the input 1^n and runs $(ssk, spk) \leftarrow \mathsf{SKGen}(1^n)$ to create a key pair for the signature scheme, and then also invokes $k \leftarrow \mathsf{KGen}_{prf}(1^n)$ to derive a key for the pseudorandom function. It outputs $(sk_{sig}, pk_{sig}) = ((ssk, k), spk)$. Algorithm $\mathsf{KGen}_{san}(1^n)$ generates a key pair $(sk_{san}, pk_{san}) = (gsk_{san}, gpk_{san}) \leftarrow \mathsf{UKGen}(1^n)$ of the group signature scheme.*

Signing. *Algorithm Sign on input $m \in \{0,1\}^*$, $sk_{sig} = (ssk, k)$, pk_{san}, ADM sets $m_{\mathrm{FIX}} = \mathrm{FIX}_{\mathrm{ADM}}(m)$ for the algorithm $\mathrm{FIX}_{\mathrm{ADM}}$ determined by ADM. It runs the user key generation algorithm $(gsk_{sig}, gpk_{sig}) \leftarrow \mathsf{UKGen}(1^n; \mathsf{PRF}(k, 0\|pk_{san}))$ for randomness $\mathsf{PRF}(k, 0\|pk_{san})$ and afterwards the group manager algorithm to compute*

$$(gmsk, gpk, cert_{sig}, cert_{san}) \leftarrow \mathsf{GKGen}(1^n, (gpk_{sig}, pk_{san}); \mathsf{PRF}(k, 1\|pk_{san}))$$

for randomness $\mathsf{PRF}(k, 1\|pk_{san})$. It computes

$$\sigma_{\mathrm{FIX}} = \mathsf{SSign}(ssk, (m_{\mathrm{FIX}}, \mathrm{ADM}, pk_{san}, gpk)) \text{ and}$$

$$\sigma_{\mathrm{FULL}} = \mathsf{GSig}(gsk_{sig}, cert_{sig}, (m, pk_{sig}), gpk)$$

using the signing algorithms of the regular and of the group signature scheme. The algorithm finally returns $\sigma = (\sigma_{\mathrm{FIX}}, \sigma_{\mathrm{FULL}}, \mathrm{ADM}, pk_{san}, cert_{san}, gpk)$.

Sanitizing. *Algorithm Sanit on input a message m, information MOD, a signature $\sigma = (\sigma_{\mathrm{FIX}}, \sigma_{\mathrm{FULL}}, \mathrm{ADM}, pk_{san}, cert_{san}, gpk)$, keys pk_{sig} and sk_{san} first recovers $m_{\mathrm{FIX}} = \mathrm{FIX}_{\mathrm{ADM}}(m)$. It then checks that MOD is admissible according to ADM and that σ_{FIX} is a valid signature for message $(m_{\mathrm{FIX}}, \mathrm{ADM}, pk_{san}, gpk)$ under key spk. If not, it stops outputting $\perp$. Else, it derives the modified message $m' = \mathrm{MOD}(m)$ and computes*

$$\sigma'_{\mathrm{FULL}} = \mathsf{GSig}(gsk_{san}, cert_{san}, (m', pk_{sig}), gpk)$$

and outputs m' together with $\sigma' = (\sigma_{\text{FIX}}, \sigma'_{\text{FULL}}, \text{ADM}, pk_{san}, cert_{san}, gpk)$.

VERIFICATION. *Algorithm* **Verify** *gets as input a message $m \in \{0,1\}^*$, a signature $\sigma = (\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM}, pk_{san}, cert_{san}, gpk)$ and public keys $pk_{sig} = spk$ and pk_{san}. It first recovers $m_{\text{FIX}} = \text{FIX}_{\text{ADM}}(m)$. It then checks whether $\mathsf{SVf}(spk, (m_{\text{FIX}}, \text{ADM}, pk_{san}, gpk), \sigma_{\text{FIX}}) = 1$ and $\mathsf{GVf}(gpk, (m, pk_{sig}), \sigma_{\text{FULL}})$ returns* `true`*, too. If so, it outputs 1, declaring the entire signature as valid. Otherwise it returns 0, indicating an invalid signature.*

PROOF. *Algorithm* **Proof** *gets as input sk_{sig}, m and $\sigma = (\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM}, pk_{san}, cert_{san}, gpk)$. It parses the key as $sk_{sig} = (ssk, k)$ and recomputes*

$$(gmsk, gpk', cert'_{sig}, cert'_{san}) = \mathsf{GKGen}(1^n, (gpk_{sig}, pk_{san}); \mathsf{PRF}(k, 1||pk_{san}))$$

and checks that $gpk' = gpk$ and $cert'_{san} = cert_{san}$ (and immediately returns $\perp$ if not). It next verifies that $\mathsf{SVf}(spk, (m_{\text{FIX}}, \text{ADM}, pk_{san}, gpk), \sigma_{\text{FIX}}) = 1$ and, if so, computes and outputs $(i, \pi) \leftarrow \mathsf{Open}(gmsk, (m, pk_{sig}), \sigma_{\text{FULL}}, gpk)$, where $i \in \{\mathtt{Sig}, \mathtt{San}\}$ is the identity returned by the **Open** *algorithm (or,* **Proof** *returns $\perp$ if any of the verification steps above fail).*

JUDGE. *The judge on input $m, \sigma, pk_{sig}, pk_{san}$ and a proof (i, π) with $i \in \{\mathtt{Sig}, \mathtt{San}\}$ parses σ as $(\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM}, pk_{san}, cert_{san}, gpk)$. It derives $b \leftarrow \mathsf{GJudge}((m, pk_{sig}), \sigma_{\text{FULL}}, i, \pi, gpk)$ using the judge algorithm of the group signature scheme. If $b =$* `true` *it outputs i, else it outputs $i = \mathtt{Sig}$.*

Completeness of signatures generated by the signer and sanitizer follows easily from the completeness of the underlying signature schemes and the fact that FIX_{ADM} leaves the fixed message parts unchanged for modified messages. There is a negligible probability that a signature of the signer or the sanitizer also verifies under the other party's other key, yielding possibly a wrong answer from the judge. We ignore this issue here for simplicity.

4.3 Security Proof

We need an additional property of the admissible modifications ADM: given arbitrary messages $m, m^* \in \{0,1\}^*$ (and a security parameter 1^n) it should be efficiently decidable whether $m^* \in \{\text{MOD}(m) \mid \text{MOD} \text{ with } \text{ADM}(\text{MOD}) = 1\}$ or not. We call such ADM *modification-decidable* and a sanitizable signature scheme *modification-restricted* if it only allows modification-decidable ADM. As an example consider again block-divided messages where ADM describes the block-length and the indices of changeable blocks. Then it is easy to check whether m^* has been changed in admissible blocks only or not.

Theorem 4.7 *Let $\mathcal{S}$ be a strongly unforgeable deterministic signature scheme and let GS be a secure group signature scheme. Assume further that $\mathcal{PRF}$ is a pseudorandom function. Then the modification-restricted sanitizable signature scheme in Construction 4.6 is unforgeable, immutable, private, proof-restricted transparent, accountable and unlinkable.*

Proof. Recall that it suffices to show immutability, proof-restricted transparency, accountability and unlinkability. The other properties then follow. Before doing so we slightly modify the scheme in the sense that we generate truly random and independent data $(gmsk, gpk, cert_{\text{sig}}, cert_{\text{san}}) \leftarrow \mathsf{GKGen}(1^n, \text{gpk}_{\text{sig}}, pk_{\text{san}})$ for each sanitizer key appearing in signature requests in the experiments.

We then maintain a list of the assigned data to keys instead of invoking the pseudorandom function each time.

By the security of the pseudorandom function it follows that the modified scheme is as a secure as the original scheme (except for a negligible security loss for each property). Here we use the fact that for each of the experiment we can decide efficiently whether the adversary has won or not, thus enabling us to turn any significantly changing success probabilities in any of the experiments into a successful distinguisher against the pseudorandom function. For all properties except for immutability this decidability is clear, for immutability it follows from the modification-restriction of the scheme.

Immutability. Recall that immutability refers to the fact that the adversary with oracle access to the signer and the proof algorithm cannot create a valid message-signature for a key pk^*_{san} such that for all pevious queries i we either have $pk^*_{\text{san}} \neq pk_{\text{san}i}$ or $m^* \notin \{\textsc{mod}(m) \mid \textsc{mod} \text{ with } \textsc{adm}_i(\textsc{mod}) = 1\}$.

Assume towards contradiction that our (modified) scheme is *not* immutable. We then show that this violates the unforgeability of the signer's signature scheme $\mathcal{S}$, i.e. we transform a successful attacker against immutability into a successful adversary against unforgeability of the signature scheme. Let $\mathcal{A}_{\text{immu}}$ be an efficient successful attacker against immutability. Given $\mathcal{A}_{\text{immu}}$ we build a successful forger $\mathcal{B}_{\text{unf}}$ against the signer's signature scheme. To this end, $\mathcal{B}_{\text{unf}}$ immitates $\mathcal{A}_{\text{immu}}$'s environment with the help of a signing oracle as follows. $\mathcal{B}_{\text{unf}}$ receives as input a public key *spk* of the signature scheme and picks the other key pairs herself. (Note that $\mathcal{B}_{\text{unf}}$ does not need to generate a key k for the pseudorandom function as we have already eliminated this part.)

Adversary $\mathcal{B}_{\text{unf}}$ then invokes $\mathcal{A}_{\text{immu}}$ on the public keys and answers each oracle query of $\mathcal{A}_{\text{immu}}$ by using the corresponding secret keys and by calling the signature oracle whenever the signer was supposed to compute a signature with the help of *ssk*. When $\mathcal{A}_{\text{immu}}$ eventually outputs pk^*_{san}, m^* and $\sigma^* = (\sigma^*_{\text{FIX}}, \sigma^*_{\text{FULL}}, \textsc{adm}^*, pk^*_{\text{san}}, cert^*_{\text{san}}, gpk^*)$. Let m^*_{FIX} be the fixed part of message m^* with respect to $\textsc{adm}^*$. Then our adversary $\mathcal{B}_{\text{unf}}$ returns the message $(m^*_{\text{FIX}}, \textsc{adm}^*, pk^*_{\text{san}}, gpk^*)$ and the forgery attempt σ^*_{FIX}.

Note that, if $\mathcal{A}_{\text{immu}}$ succeeds, it must particularly hold that σ^*_{FIX} is a valid signature for m^*_{FIX}. Hence, it suffices to show that the message m^*_{FIX} has not been submitted by $\mathcal{B}_{\text{unf}}$ to the signature oracle before. If this was the case then it must have happened for a signature request (else $\mathcal{B}_{\text{unf}}$ does not call the external signing oracle) for some signature query i. But then it must be that $pk^*_{\text{san}} = pk_{\text{san}i}$ and $\textsc{adm}^* = \textsc{adm}_i$ for this query, and thus – as $\mathcal{A}_{\text{immu}}$ succeeds – we have $m^* \notin \{\textsc{mod}(m_i) \mid \textsc{mod} \text{ with } \textsc{adm}_i(\textsc{mod}) = 1\}$. And thus, by the maximality requirement of $\textsc{fix}_{\textsc{adm}}$ (see subsection 2.1), it follows that $\textsc{fix}_{\textsc{adm}^*}(m^*) \neq \textsc{fix}_{\textsc{adm},i}(m_i)$, i.e., $m_{\textsc{fix}}{}^* \neq m_{\textsc{fix},i}$ and the output message $(m^*_{\text{FIX}}, \textsc{adm}^*, pk^*_{\text{san}}, gpk^*)$ with signature σ^*_{FIX} thus constitutes a valid forgery for our adversary $\mathcal{B}_{\text{unf}}$.

Sanitizer-Accountabiliy. Remember that sanitizer-accountability guarantees that for a malicious sanitizer, having access to a signing oracle Sign and to a proof oracle Proof, it is infeasible to create a fresh triple consisting of a valid message-signature pair (m^*, σ^*) and a sanitizer key pk^*_{san} such that the proof π, provided by the signer's Proof algorithm, makes the judge to accuse the signer.

Assume towards contradiction that $\mathcal{A}_{\text{san}}$ is an efficient attacker and breaks sanitizer-accountability. We then show how to build an algorithm $\mathcal{B}_{\text{trace}}$ against the traceability of the group signature scheme GS. The input of the algorithm $\mathcal{B}_{\text{trace}}$, initialized in mode `choose`, is a public key $pk_{\text{user},0}$. She first generates a key pair $(ssk, spk) \leftarrow \mathsf{SKGen}(1^n)$ of the underlying deterministic signature scheme $\mathcal{S}$ and simulates $\mathcal{A}_{\mathrm{san}}$ on input $pk_{\mathrm{sig}} = spk$ in a black-box way. In the following let q denote $\mathcal{A}_{\mathrm{san}}$'s maximal number of signature and proof queries to oracles Sign and Proof, and let Q be an initially empty list in which $\mathcal{B}_{\mathrm{trace}}$ stores tuples of the form $(pk_{\mathrm{san}}, (\mathrm{gsk}_{\mathrm{sig}}, \mathrm{gpk}_{\mathrm{sig}}, gmsk, gpk, cert_{\mathrm{sig}}, cert_{\mathrm{san}}))$.

$\mathcal{A}_{\mathrm{san}}$'s output is a triple $(pk^*_{\mathrm{san}}, m^*, \sigma^*)$, and algorithm $\mathcal{B}_{\mathrm{trace}}$ tries to guess the first $\mathcal{A}_{\mathrm{san}}$ execution in which $\mathcal{A}_{\mathrm{san}}$ sends the "output" sanitizer public-key pk^*_{san} to her signing oracle *or* the Proof-oracle, and also whether or not this happens for a signature or proof query. Here, we refer to the $(q + 1)$-st query as the case that the sanitizer's public key pk^*_{san} has never been sent to the signing oracle but only appears in the output. More formally, $\mathcal{B}_{\mathrm{trace}}$ picks an index $k \in \{1, \ldots, q+1\}$ uniformly at random and a bit $p \leftarrow \{0, 1\}$ such that $pk_{\mathrm{san},k}$ is our "target" key of the sanitizer in a signature request ($p = 0$) or a proof request ($p = 1$). We note that algorithm $\mathcal{B}_{\mathrm{trace}}$ will start the `guess` stage with the k-th query and stays in mode `choose` till then.

We now turn to the simulation of the signing oracle Sign and of the proof oracle Proof. We distinguish between two cases. Firstly, we consider the case where the current number j of the query to the Sign-oracle equals our guess k and we predict that the target key appears for the first time in a signature query, i.e., $j = k$ and $p = 0$. Secondly, we deal with the case where $j \neq k$ or $p = 1$.

CASE 1: $j = k$ AND $p = 0$. Let assume that the adversary $\mathcal{A}_{\mathrm{san}}$ queries the signing oracle Sign for the k-th time on a message m, on a sanitizer's public key pk_{san}, and on a description of admissibly modifiable parts ADM. In this particular case, $\mathcal{B}_{\mathrm{trace}}$ stores $((ssk, spk), Q)$ in her state st and outputs (st, pk_{san}). Afterwards, $\mathcal{B}_{\mathrm{trace}}$ is initialized on input $(st, gpk, \mathbf{cert})$ in mode `guess` and has access to a group signature oracle GSig and to an open oracle Open. Adversary $\mathcal{B}_{\mathrm{trace}}$ parses the state st as $((ssk, spk), Q)$ and answers the k-th query now as follows: She sets $m_{\mathrm{FIX}} = \mathrm{FIX}_{\mathrm{ADM}}(m)$ for the algorithm $\mathrm{FIX}_{\mathrm{ADM}}$ determined by ADM, computes $\sigma_{\mathrm{FIX}} = \mathsf{SSign}(ssk, (m_{\mathrm{FIX}}, \mathrm{ADM}, pk_{\mathrm{san}}, gpk))$ and invokes her external group signature GSig oracle on message (m, pk_{sig}). The oracle returns the group signature σ_{FULL}. Algorithm $\mathcal{B}_{\mathrm{trace}}$ returns the signature $\sigma = (\sigma_{\mathrm{FIX}}, \sigma_{\mathrm{FULL}}, \mathrm{ADM}, pk_{\mathrm{san}}, cert_{\mathrm{san}}, gpk)$ to adversary $\mathcal{A}_{\mathrm{san}}$.

CASE 2: $j \neq k$ OR $p = 1$. Whenever $\mathcal{A}_{\mathrm{san}}$ invokes her signing oracle Sign on a tuple $(m, pk_{\mathrm{san}}, \mathrm{ADM})$ for $j \neq k$, then $\mathcal{B}_{\mathrm{trace}}$ (either still in mode `choose` or already in mode `guess`) computes the answer as follows.

If $j > k$ and $pk_{\mathrm{san},j} = pk_{\mathrm{san},k}$ then $\mathcal{B}_{\mathrm{trace}}$ (working in mode `guess`) sets $m_{\mathrm{FIX}} = \mathrm{FIX}_{\mathrm{ADM}}(m)$ for the algorithm $\mathrm{FIX}_{\mathrm{ADM}}$ determined by ADM, computes $\sigma_{\mathrm{FIX}} = \mathsf{SSign}(ssk, (m_{\mathrm{FIX}}, \mathrm{ADM}, pk_{\mathrm{san}}, gpk))$ and invokes her external group signature oracle on input $(cert_{\mathrm{sig}}, (m, pk_{\mathrm{sig}}))$, which returns the group signature σ_{FULL}. Algorithm $\mathcal{B}_{\mathrm{trace}}$ returns the signature $\sigma = (\sigma_{\mathrm{FIX}}, \sigma_{\mathrm{FULL}}, \mathrm{ADM}, pk_{\mathrm{san}}, cert_{\mathrm{san}}, gpk)$ to $\mathcal{A}_{\mathrm{san}}$.

Else, if $j < k$ or $pk_{\mathrm{san},j} \neq pk_{\mathrm{san},k}$, then adversary $\mathcal{B}_{\mathrm{trace}}$ sets $m_{\mathrm{FIX}} = \mathrm{FIX}_{\mathrm{ADM}}(m)$ for the algorithm $\mathrm{FIX}_{\mathrm{ADM}}$ determined by ADM. She checks if $pk_{\mathrm{san},j}$ is stored in Q, and if so, then she recovers the stored keys from the list Q, denoted by $(\mathrm{gsk}_{\mathrm{sig}}, \mathrm{gpk}_{\mathrm{sig}}, gmsk, gpk, cert_{\mathrm{sig}}, cert_{\mathrm{san}})$, and computes the signature

$$\sigma_{\mathrm{FIX}} = \mathsf{SSign}(ssk, (m_{\mathrm{FIX}}, \mathrm{ADM}, pk_{\mathrm{san}}, gpk)) \text{ and } \sigma_{\mathrm{FULL}} = \mathsf{GSig}(\mathrm{gsk}_{\mathrm{sig}}, cert_{\mathrm{sig}}, (m, pk_{\mathrm{sig}}), gpk)\,.$$

Otherwise, if $pk_{\mathrm{san},j}$ is not stored in Q, then $\mathcal{B}_{\mathrm{trace}}$ runs the key generation of the user $(\mathrm{gsk}_{\mathrm{sig}}, \mathrm{gpk}_{\mathrm{sig}}) \leftarrow \mathsf{UKGen}(1^n)$ as well as the group manager algorithm $\mathsf{GKGen}(1^n, (\mathrm{gpk}_{\mathrm{sig}}, pk_{\mathrm{san}}))$ in order to derive a fresh key $(gmsk, gpk, cert_{\text{sig}}, cert_{\text{san}})$. Algorithm $\mathcal{B}_{\text{trace}}$ stores the tuple $(pk_{\text{san},j}, (\text{gsk}_{\text{sig}}, \text{gpk}_{\text{sig}}, gmsk, gpk, cert_{\text{sig}}, cert_{\text{san}}))$ in Q and computes the signature

$$\sigma_{\text{FIX}} = \mathsf{SSign}(ssk, (m_{\text{FIX}}, \text{ADM}, pk_{\text{san}}, gpk)) \text{ and } \sigma_{\text{FULL}} = \mathsf{GSig}(\text{gsk}_{\text{sig}}, cert_{\text{sig}}, (m, pk_{\text{sig}}), gpk)$$

using the signing algorithms of the regular and of the group signature scheme and returns $\sigma = (\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM}, pk_{\text{san}}, cert_{\text{san}}, gpk)$.

We now turn to the simulation of the proof oracle Proof. We again distinguish between the cases that our alleged target key shows up in the j-th query to the Proof oracle and $p = 1$, and the other cases.

Case 1: $j = k$ and $p = 1$. At this point $\mathcal{B}_{\text{trace}}$ is still in mode `choose`. Suppose adversary $\mathcal{A}_{\text{san}}$ has sent a message m and on a signature $\sigma = (\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM}, pk_{\text{san}}, cert_{\text{san}}, gpk)$ to the oracle. Then $\mathcal{B}_{\text{trace}}$ outputs (st, pk_{san}) for state $st = ((ssk, spk), Q)$ and changes into mode `guess`. It receives $(st, gpk, \textsf{\textbf{cert}})$ as input and answers the query as follows. She first runs the same verification checks as the Proof-oracle (comparing the public key data and verifiying the signature's validity). Adversary $\mathcal{B}_{\text{trace}}$ next invokes her external open oracle Open on the pair $(m, \sigma_{\text{FULL}})$ and returns the answer (i, π).

Case 2: $j \neq k$ or $p = 0$. In order to answer such a signature query, we have to distinguish between two cases. Firstly, in the case that $pk_{\text{san},j} \neq pk_{\text{san},k}$ or $j < k$, $\mathcal{B}_{\text{trace}}$ checks whether $pk_{\text{san},j}$ is in the list Q. Suppose that pk_{san} is in Q, then let $(\text{gsk}_{\text{sig}}, \text{gpk}_{\text{sig}}, gmsk, gpk, cert_{\text{sig}}, cert_{\text{san}})$ be the corresponding keys. $\mathcal{B}_{\text{trace}}$ then performs the same steps as the Proof-oracle and returns (i, π). Otherwise, if $pk_{\text{san},j} \neq pk_{\text{san},k}$ and $pk_{\text{san},j}$ is not in Q, then $\mathcal{B}_{\text{trace}}$ generates fresh keys:

$$(gmsk, gpk, cert_{\text{sig}}, cert_{\text{san}}) = \mathsf{GKGen}(1^n, (\text{gpk}_{\text{sig}}, pk_{\text{san}}))$$

and proceeds as the Proof-oracle to return (i, π). In the case that $pk_{\text{san},j} = pk_{\text{san},k}$, then $\mathcal{B}_{\text{trace}}$ invokes her external open oracle Open on the pair $(m, \sigma_{\text{FULL}})$ and returns the answer (i, π) (if the correctness checks that Proof would perform also succeed).

Finally, $\mathcal{A}_{\text{san}}$ stops, outputting a tuple $(pk^*_{\text{san}}, m^*, \sigma^*)$. σ^* contains a group signature σ^*_{FULL} over the message $m^*_{\text{FULL}} = (m^*, pk_{\text{sig}})$. $\mathcal{B}_{\text{trace}}$ returns the pair $(m^*_{\text{FULL}}, \sigma^*_{\text{FULL}})$ as her final output. This concludes the description of the simulation.

Analysis. For the analysis first observe that $\mathcal{B}_{\text{trace}}$ is efficient because $\mathcal{A}_{\text{san}}$ runs in polynomial-time and the additional steps can all be carried out efficiently. Given that we predict the first appearance of pk^*_{san} correctly, which happens with probability at least $\frac{1}{2(q+1)}$ for a polynomial q, $\mathcal{B}_{\text{trace}}$ performs a perfect simulation from $\mathcal{A}_{\text{san}}$'s point of view. Let assume that $\mathcal{A}_{\text{san}}$ succeeds with noticeable probability $\epsilon(n)$ and

Whenever $\mathcal{A}_{\text{san}}$ succeeds, then she outputs a valid triple $(pk^*_{\text{san}}, m^*, \sigma^*)$ such that for the pair (m^*, pk^*_{san}) the common freshness condition holds, i.e., $\mathcal{A}_{\text{san}}$ has never queried her signing oracle Sign about this pair. But if the tuple is fresh and $\mathcal{B}_{\text{trace}}$ has guessed the right target key of the sanitizer, then all queries forwarded by $\mathcal{B}_{\text{trace}}$ to the external signing oracle have been for this key pk^*_{san} and for different messages than m^*_{FULL} (else (m^*, pk^*_{san}) would have been a previous request by $\mathcal{A}_{\text{san}}$). Hence, the tuple $(m^*_{\text{FULL}}, \sigma^*_{\text{FULL}}, pk^*_{\text{san}})$ is also a valid output in the sense of the traceability experiment (Note that $pk^*_{\text{san}} = \text{gpk}^*_{\text{san}}$).

It remains to show that if $\mathcal{A}_{\text{san}}$'s output $(pk^*_{\text{san}}, m^*, \sigma^*)$ is a successful attack, then the triple $(m^*_{\text{FULL}}, \sigma^*_{\text{FULL}}, pk^*_{\text{san}})$ satisfies the success conditions for an attacker against traceability. If $\mathcal{A}_{\text{san}}$ is successful, then we have that $\mathsf{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk^*_{\text{san}}) = \mathtt{true}$ and $\mathsf{Judge}(m^*, \sigma^*, pk_{\text{sig}}, pk^*_{\text{san}}, \pi) =$ sig for a genuinely generated proof Π. That means, that $\mathsf{GVf}(gpk, m^*_{\text{FULL}}, \sigma^*_{\text{FULL}}) = \mathtt{true}$ and either it holds that $\mathsf{Open}(gmsk, m^*_{\text{FULL}}, \sigma_{\text{FULL}}, gpk) = (i, \pi) = \perp$ either for (i, π), we have $i = 0$ and $\mathsf{GJudge}(m^*_{\text{FULL}}, \sigma_{\text{FULL}}, 0, \pi, gpk) = \mathtt{true}$. Thus, any successful attack on the sanitizer-accountability yields a successful attack to the traceability of the underlying group signature scheme.

Signer-Accountability. Let $\mathcal{A}_{\text{sig}}$ successfully break the signer-accountability. We then derive from $\mathcal{A}_{\text{sig}}$ an attacker $\mathcal{B}_{\text{nonfr}}$ against the non-frameability property of the group signature scheme. $\mathcal{B}_{\text{nonfr}}$ gets as input the public key $pk_{\text{user},1}$ and has access to a GSig-oracle endowed with the secret key $sk_{\text{user},1}$. At first, $\mathcal{B}_{\text{nonfr}}$ forwards $pk_{\text{user},1}$ to $\mathcal{A}_{\text{sig}}$ as the sanitizer's public key. When $\mathcal{A}_{\text{sig}}$ issues a query $(m_i, \sigma_i, \text{MOD}_i, pk_{\text{sig,i}})$ to her Sanit oracle, $\mathcal{B}_{\text{nonfr}}$ executes the same checks on these inputs, as the sanitizing algorithm would do. If all these checks succeed, $\mathcal{B}_{\text{nonfr}}$ parses σ_i into its components in order to deduce $\mathsf{cert}_{san,i}$ and gpk_i. She then computes $m'_i = \text{MOD}_i(m_i)$ and queries $(\mathsf{cert}_{san,i}, (m'_i, pk_{\text{sig,i}}), gpk_i)$ to her GSig oracle which returns a signature $\sigma'_{\text{FULL},i}$. $\mathcal{B}_{\text{nonfr}}$ then obtains σ'_i from σ_i by replacing $\sigma_{\text{FULL},i}$ by $\sigma'_{\text{FULL},i}$. She returns σ'_i to $\mathcal{A}_{\text{sig}}$. Note that this simulation is perfect from $\mathcal{A}_{\text{sig}}$'s point of view. Finally, $\mathcal{A}_{\text{sig}}$ outputs a quadruple $(pk^*_{\text{sig}}, m^*, \sigma^*, \pi^*)$. $\mathcal{B}_{\text{nonfr}}$ parses σ^* into its components and derives σ^*_{FULL}, gpk^* and $\mathsf{cert}^*_{\text{san}}$. We now argue that if $\mathcal{A}_{\text{sig}}$ is successful, then $\mathcal{B}_{\text{nonfr}}$ satisfies the success condition of the non-frameability game by outputting $((m^*, pk^*_{\text{sig}}), \sigma^*_{\text{FULL}}, \pi^*, gpk^*, \mathsf{cert}^*_{\text{san}})$. First of all, if $\mathcal{A}_{\text{sig}}$ wins, then one has $\mathsf{Verify}(m^*, \sigma^*, pk^*_{\text{sig}}, pk_{\text{san}}) = \mathtt{true}$. Thus, σ^*_{FULL} is a valid group signature over (m^*, pk_{sig}) under group public key gpk^*. Moreover, we have that $(pk^*_{\text{sig}}, m^*) \neq (pk_{\text{sig,i}}, m'_i)$ for all previous queries $(m_i, \sigma_i, \text{MOD}_i, pk_{\text{sig,i}})$ to the Sanit oracle. This implies that during the simulation, $\mathcal{B}_{\text{nonfr}}$ does not query the GSig-oracle about message (m^*, pk^*_{sig}). In addition, it holds that $\mathsf{Judge}(m^*, \sigma^*, pk^*_{\text{sig}}, pk_{\text{san}}, \pi^*)$ outputs $\mathtt{San}$, which implies that $\mathsf{GJudge}(m^*, \sigma^*, 1, \pi^*, gpk^*)$ outputs $\mathtt{true}$. Thus, the success probability of $\mathcal{B}_{\text{nonfr}}$ equals the success probability of $\mathcal{A}_{\text{sig}}$.

(Proof-Restricted) Transparency. Transparency says that it is infeasible for any efficient adversary $\mathcal{A}_{\text{trans}}$ to find out whether a signed message was issued by the signer or by the sanitizer. This should even hold if $\mathcal{A}_{\text{trans}}$ accesses a signer- and a sanitizer-oracle as well as a SanSig-oracle which, depending on a bit b, either always responds with sanitizer outputs or either always returns signer outputs. Additionally, we even allow $\mathcal{A}_{\text{trans}}$ to use Proof in order to find out about a message's origin (but in the proof-restricted case the adversary is not allowed to query Proof for outputs of $\mathsf{Sanit}/\mathsf{Sign}$ to avoid trivial attacks).

We assume towards contradiction that our scheme is not proof-restricted transparent. We prove that in this case, anonymity of the underlying group signature scheme cannot hold. Let $\mathcal{A}_{\text{trans}}$ be a successful adversary against transparency, from which we build adversary $\mathcal{B}_{\text{anon}}$, a successful attacker against the anonymity of the underlying group signature scheme. Algorithm $\mathcal{B}_{\text{anon}}$ simulates $\mathcal{A}_{\text{trans}}$'s oracles as described next.

$\mathcal{B}_{\text{anon}}$ first generates keys (ssk, spk) for SSign. She gets $gpk, sk_{\text{user}0}, sk_{\text{user}1}, \mathsf{cert}_0, \mathsf{cert}_1$ as inputs, which enables her to sign and sanitize messages on behalf of the signer and the sanitizer (possibly picking fresh group keys for signature queries involving a different sanitizer public key than $pk_{\text{user},1}$). Thus, she can easily answer $\mathcal{A}_{\text{trans}}$'s request to the Sign-oracle as well as to the Sanit-oracle.

Requests $(m_k, \text{MOD}_k, \text{ADM}_k)$ of $\mathcal{A}_{\text{trans}}$ to $\mathsf{Sanit/Sign}$ are answered as follows: $\mathcal{B}_{\text{anon}}$ signs the message $(m_{\text{FIX}k}, \text{ADM}_k, pk_{\text{san}}, gpk)$ via SSign thus producing $\sigma_{\text{FIX}k}$. Then, she passes the message $(m_k, pk_{\text{sig},k})$ to her $\mathsf{LoRSign}$-oracle and gets a group signature σ_k over it, which is either a signer signature (user 0) or a sanitizer signature (user 1). Algorithm $\mathcal{B}_{\text{anon}}$ returns $(\sigma_{\text{FIX}k}, \sigma_k, \text{ADM}_k, pk_{\text{san}}, cert_{\text{san}}, gpk)$ to $\mathcal{A}_{\text{trans}}$. Similarly, if $\mathcal{A}_{\text{trans}}$ sends a request $(m_i, \sigma_{\text{FIX}i}, \sigma_{\text{FULL}i}, \text{ADM}_i, pk_{\text{san},i}, cert_{\text{san},i}, gpk_i)$ to oracle Proof, then, if $gpk = gpk_i$, algorithm $\mathcal{B}_{\text{anon}}$ inputs $(m_i, pk_{\text{sig},i}), \sigma_{\text{FULL}i}$ into Open to receive (d_i, π_i) with $d_i \in \{0, 1\}$ (or $\perp$); she passes (d_i, π_i) to $\mathcal{A}_{\text{trans}}$. For $gpk \neq gpk_i$ she simply returns $\perp$. At the end $\mathcal{A}_{\text{trans}}$ outputs a bit a which $\mathcal{B}_{\text{anon}}$ copies and stops.

For the analysis note that the simulation is perfect in the sense that all simulated oracle replies are distributed as in the genuine attack. Hence, it remains to show that the restrictions on the anonymity adversary apply. Note that, if $\mathcal{A}_{\text{trans}}$ is successful, she has not queried messages m to Proof which have been previous outputs of oracle $\mathsf{Sanit/Sign}$. But then it follows immediately that $\mathcal{B}_{\text{anon}}$ has neither queried pairs (m, pk_{sig}) to Open that have been output by $\mathsf{LoRSign}$.

Thus, if $\mathcal{A}_{\text{trans}}$ is successful then so is $\mathcal{B}_{\text{anon}}$.

Unlinkability. In this section we show that our sanitizable signature is unlinkable, i.e., given access to a signing oracle, a sanitizing oracle and a proof oracle an adversary cannot essentially distinguish left or right outputs of an oracle $\mathsf{LoRSanit}$ better than by flipping a coin. The oracle $\mathsf{LoRSanit}$, initialized with a random bit b, takes two pairs $(m_0, \text{MOD}_0, \sigma_0, \text{ADM}_0)$ and $(m_1, \text{MOD}_1, \sigma_1, \text{ADM}_1)$ such that $\text{ADM}_0 = \text{ADM}_1$, both signatures are valid and the modified message $\text{MOD}_0(m_0) = \text{MOD}_1(m_1)$ coincide. It outputs $\mathsf{Sanit}(m_b, \text{MOD}_b, \sigma_b, pk_{\text{sig}}, sk_{\text{san}})$.

First note that for each query to $\mathsf{LoRSanit}$ we have $\text{ADM} := \text{ADM}_0 = \text{ADM}_1$ and since admissible modifications do not change the fixed part of messages and the modified messages of a query coincide, it also holds

$$m_{\text{FIX},0} = \text{FIX}_{\text{ADM}}(m_0) = \text{FIX}_{\text{ADM}}(\text{MOD}_0(m_0)) = \text{FIX}_{\text{ADM}}(\text{MOD}_1(m_1)) = \text{FIX}_{\text{ADM}}(m_1) = m_{\text{FIX},1}.$$

Hence, we must have that the signature component for the fixed parts must be the same for both messages. If an adversary would submit distinct signatures $\sigma_{\text{FIX}0} \neq \sigma_{\text{FIX}1}$ for these two parts and both would be valid, it would straightforwardly contradict the strong unforgeability of the regular signature scheme. Namely, since the honest signer produces those signatures deterministically it has only output one of the two signatures so far (collisions among different messages would immediately imply a forgery and cannot happen with more than negligible probability). Hence, such a query with identical messages and distinct signatures for the fixed part must thus contain a forgery.

We conclude that the signature parts σ_{FIX} for the sanitized messages must be identical. Since we also include the group manager's public key for both values $b \in \{0, 1\}$ the signature for the full message is a group signature for the same sanitized message $m'_0 = m'_1$ under the same group key and group member key, computed from scratch and thus identically distributed in both cases. It follows that, unless the adversary creates a forgery against the signer's signature scheme, the probability of predicting b is exactly $\frac{1}{2}$. □

5 Variations

In this section we briefly discuss two variations of our generic construction described in Section 4.2. First we show that we can achieve a more efficient construction when we drop the accountability property and then we explain how to extend our construction to address multiple sanitizers.

5.1 Construction Based on Ring Signatures

Our construction based on group signatures utilized the traceability and non-frameability property of the underlying group signature scheme to ensure accountability of the sanitizable signature scheme. If one is willing to sacrifice the accountability property, we can replace the group signature with a ring signature, yet obtaining a significantly more efficient construction.

Ring signatures were introduced in [RST01] and can be seen as a weaker variant of group signatures. That is, a member of the ring is still able to sign a message on behalf of the ring without revealing its identity, but in contrast to group signatures the rings do not require any central group manager and can be formed in an ad-hoc manner. However, due to the absence of a designated group manager, ring signatures do not enjoy the traceability property, i.e., it is not possible to revoke the anonymity of a signer for a particular message. Thus, on the one hand ring signatures comes with less security features, but on the other hand allows greater flexibility.

In the context of our sanitizable signature construction we can observe two benefits from using rings instead of groups. First, the use of ring signatures eliminates the issuing of certificates. Second, by limiting rings to only two group members, i.e., requiring a unique ring for each signer/sanitizer pair, we can apply a 2-user ring signature scheme for which very efficient constructions in the standard model exist [BKM06]. By the anonymity and unforgeability property of ring signatures, the proofs of transparency, unlinkability and immutability of our group based sanitizable signature scheme carries over to the ring based version. Unforgeability, which needs to be shown from scratch now, follows easily from the unforgeability of both, the regular and ring signature scheme.

5.2 Multiple Sanitizers

The proposed construction only considers the case of a single sanitizer. However, as group signatures allow groups for more than two member, we can easily extend our construction to become applicable for multiple sanitizer. To this end, the signer simply uses the group signature for a group that now contains (possibly) multiple sanitizers. Later, the signer can partially delegate its signing rights to several sanitizers of the group, by including the public keys of all designated sanitizers into the signature of the fixed part of the message.

In this setting, it might by likely that neither the number nor the identities of all sanitizers are known in the key generation phase. (Partially) dynamic group signature schemes [BSZ05] capture exactly that situation, i.e., the group manager only chooses a group public key in the setup phase and subsequently an entity (i.e., sanitizer) can join the group by engaging in a join protocol with the group manager. Dynamic group signature schemes, that meet our requirements are proposed e.g., in [BSZ05, DP06].

Acknowledgments

We thank the anonymous reviewers for valuable comments. Marc Fischlin, Anja Lehmann and Dominique Schröder are supported by the Emmy Noether Program Fi 940/2-1 of the German Research Foundation (DFG). This work was also supported by CASED (`www.cased.de`).

References

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

[BB04] Dan Boneh and Xavier Boyen. *Short Signatures Without Random Oracles.* Advances in Cryptology — Eurocrypt'04, Volume 3027 of Lecture Notes in Computer Science, pages 56–73. Springer-Verlag, 2004.

[BFF$^+$09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schroeder, and Florian Volk. *Security of Sanitizable Signatures Revisited.* Public-Key Cryptography (PKC) 2009, Volume 5443 of Lecture Notes in Computer Science, pages 317–336. Springer-Verlag, 2009.

[BKM06] Adam Bender, Jonathan Katz, and Ruggero Morselli. *Ring Signatures: Stronger Definitions, and Constructions Without Random Oracles.* Theory of Cryptography Conference (TCC)2006, Volume 3876 of Lecture Notes in Computer Science, pages 60–79. Springer-Verlag, 2006.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. *Short Signatures from the Weil Pairing. Journal of Cryptology*, 17(4):297–319, 2004.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. *Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions.* Advances in Cryptology — Eurocrypt2003, Volume 2656 of Lecture Notes in Computer Science, pages 614–629. Springer-Verlag, 2003.

[BR96] Mihir Bellare and Phillip Rogaway. *The Exact Security of Digital Signatures - How to Sign with RSA and Rabin.* Advances in Cryptology — Eurocrypt 1996, pages 399–416. Springer-Verlag, 1996.

[BS07] Mihir Bellare and Sarah Shoup. *Two-Tier Signatures, Strongly Unforgeable Signatures, and Fiat-Shamir Without Random Oracles.* Public-Key Cryptography (PKC)'07, Lecture Notes in Computer Science, pages 201–216. Springer-Verlag, 2007.

[BSW06] Dan Boneh, Emily Shen, and Brent Waters. *Strongly Unforgeable Signatures Based on Computational Diffie-Hellman.* Public-Key Cryptography (PKC)'06, Lecture Notes in Computer Science, pages 229–240. Springer-Verlag, 2006.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. *Foundations of Group Signatures: The Case of Dynamic Groups.* CT-RSA 2005, Volume 3376 of Lecture Notes in Computer Science. Springer-Verlag, 2005.

[CJ10] Sébastien Canard and Amandine Jambert. *On Extended Sanitizable Signature Schemes.* Topics in Cryptology — Cryptographer's Track, RSA Conference (CT-RSA) 2010, Volume 5985 of Lecture Notes in Computer Science, pages 179–194. Springer-Verlag, 2010.

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

[Cor00] Jean-Sebastien Coron. *On the Exact Security of Full Domain Hash.* Advances in Cryptology — Crypto2000, Volume 1880 of Lecture Notes in Computer Science, pages 229–235. Springer-Verlag, 2000.

[CvH91] D. Chaum and E. van Heyst. *Group Signatures.* Advances in Cryptology — Eurocrypt'91, Volume 547 of Lecture Notes in Computer Science, pages 241–246. Springer-Verlag, 1991.

[DP06] Cécile Delerablée and David Pointcheval. *Dynamic Fully Anonymous Short Group Signatures.* VIETCRYPT, Volume 4341 of Lecture Notes in Computer Science, pages 193–210. Springer-Verlag, 2006.

[Gol87] Oded Goldreich. *Two Remarks Concerning the Goldwasser-Micali-Rivest Signature Scheme.* Advances in Cryptology — Crypto '86, Volume 263 of Lecture Notes in Computer Science, pages 104–110. Springer-Verlag, 1987.

[Gro06] Jens Groth. *Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures.* Advances in Cryptology — Asiacrypt, Volume 4284 of Lecture Notes in Computer Science, pages 444–459. Springer-Verlag, 2006.

[Gro07] Jens Groth. *Fully Anonymous Group Signatures Without Random Oracles.* Advances in Cryptology — Asiacrypt, Volume 4833 of Lecture Notes in Computer Science, pages 164–180. Springer-Verlag, 2007.

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

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

[KY05] Aggelos Kiayias and Moti Yung. *Group Signatures with Efficient Concurrent Join.* Advances in Cryptology — Eurocrypt 2005, Volume 3494 of Lecture Notes in Computer Science, pages 198–214. Springer-Verlag, 2005.

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

[RST01] Ronald Rivest, Adi Shamir, and Yael Tauman. *How to Leak a Secret.* Advances in Cryptology — Asiacrypt2001, Volume 2248 of Lecture Notes in Computer Science, pages 552–565. Springer-Verlag, 2001.

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

A Other Cryptographic Primitives

Signatures. A signature scheme $\mathcal{S} = (\mathsf{SKGen}, \mathsf{SSign}, \mathsf{SVf})$ is a triple of efficient algorithms such that $\mathsf{SKGen}(1^n)$ returns a key pair (ssk, spk), SSign on input ssk and a message m^* returns a signature σ, and SVf on input spk, m, σ returns a bit d. It should hold that for any $n \in \mathbb{N}$, any $(ssk, spk) \leftarrow \mathsf{SKGen}(1^n)$, any $m \in \{0,1\}^*$, any $\sigma \leftarrow \mathsf{SSign}(ssk, m)$ that $\mathsf{SVf}(spk, m, \sigma) = 1$.

The signature scheme is called *unforgeable* if for any efficient algorithm $\mathcal{B}$ the probability that, on input spk (generated as part of $(ssk, spk) \leftarrow \mathsf{SKGen}(1^n)$), and with oracle access to $\mathsf{SSign}(ssk, \cdot)$, algorithm $\mathcal{A}$ outputs m^*, σ^* such that $\mathsf{SVf}(spk, m^*, \sigma^*) = 1$ and $\mathcal{B}$ has never submitted m^* to the SSign-oracle, is negligible. It is called *strongly unforgeable* if the probability remains negligible even if the adversary only outputs (m^*, σ^*) which has never appeared in the communication with oracle SSign.

Pseudorandom Functions. A pseudorandom function $\mathcal{PRF} = (\mathsf{KGen}_{\mathrm{prf}}, \mathsf{PRF})$ is a pair of efficient algorithms with $\mathsf{PRF}(k, \cdot) : \{0,1\}^* \rightarrow \{0,1\}^n$ for $k \leftarrow \mathsf{KGen}_{\mathrm{prf}}(1^n)$ such that for any efficient algorithm $\mathcal{B}$ the following value is negligible:

$$\left|\mathrm{Prob}\left[\mathcal{B}^{\mathsf{PRF}(k,\cdot)}(1^n) = 1\right] - \mathrm{Prob}\left[\mathcal{B}^{\mathcal{R}}(1^n) = 1\right]\right|,$$

where the first probability is taken over $k \leftarrow \mathsf{KGen}_{\mathrm{prf}}(1^n)$ and $\mathcal{B}$'s internal toin cosses, and the second probability is over the choice of the random function $\mathcal{R} : \{0,1\}^* \rightarrow \{0,1\}^n$ and $\mathcal{B}$'s randomness.