Sanitizable Signatures: How to Partially Delegate Control for Authenticated Data

Christina Brzuska Marc Fischlin Anja Lehmann
Dominique Schröder

Darmstadt University of Technology, Germany
www.minicrypt.de

Abstract. Sanitizable signatures have been introduced by Ateniese et al. (ESORICS 2005) and allow an authorized party, the sanitizer, to modify a predetermined part of a signed message without invalidating the signature. Brzuska et al. (PKC 2009) gave the first comprehensive formal treatment of the five security properties for such schemes. These are unforgeability, immutability, privacy, transparency and accountability. They also provide a modification of the sanitizable signature scheme proposed by Ateniese et al. such that it provably satisfies all security requirement. Unfortunately, their scheme comes with rather large signature sizes and produces computational overhead that increases with the number of admissible modifications.

In this paper we show that by sacrificing the transparency property —thus allowing to distinguish whether a message has been sanitized or not— we can obtain a sanitizable signature scheme that is still provably secure concerning the other aforementioned properties but significantly more efficient. We propose a construction that is based solely on regular signature schemes, produces short signatures and only adds a small computational overhead.

1 Introduction

Digital signatures usually provide integrity and authenticity of digital data. This, in particular, implies that even slight modifications of the data make the signature invalid. There are, however, some cases where allowing such modifications while retaining the authenticity to a certain extend may be desirable. For example,

- Governmental organizations like the World Health Organization (WHO) may ask medical facilities to provide medical records for infectious disease surveillance programs. Allowing the administration of such facilities to sanitize parts of the records (which are authenticated by medical personal through signatures) like patient names or information about psychological treatments eases the overhead. At the same time it still marks the resulting data as authenticated by medical personal.
- Authenticated multimedia data like videos may require some editing, e.g., because of graphic content or to insert local commercials into the data.

- Authenticated routing information as in the Secure Border Gateway Protocol needs to be updated frequently, while the reliability of the data must be ensured.

As another example, consider the recent discussion about German identity cards and the digital data stored on the card [Bun08]. The data includes common information about the holder like the name, date of birth and address. These data are not signed, though, to guarantee deniability of transactions —else a party retrieving such signed data can show this as a proof for a transaction to third parties— and to possibly enable modifications by subordinate authorities to volatile data like the address (see [BKMN08]). Note that in the non-digital case local authorities today can easily change the address by placing an (authenticated) sticker on the identity card. In the digital case, any signature over the holder's data would prevent such modifications (unless the issuing authority would bequeath the signing key, which is of course not recommended).

Enter sanitizable signatures. The notion of sanitizable signatures has been introduced by Ateniese et al. [ACdMT05]. Similar notions have been considered concurrently by Steinfeld et al. [SBZ01] and Miyazaki et al. [MSI+03]. The idea behind sanitizable signatures is that the signer delegates the signing rights of parts of the message to a designated party, the sanitizer. The sanitizer, given a message and a signature of the signer, can then modify the predetermined parts of the message and still produce a valid signature for the new message. A verifier of this new signatures is then assured that (a) the fixed message parts have been authenticated by the signer, and (b) that only the designated sanitizer can make admissible modifications.

Sanitizable signature are thus an expedient solution to the scenarios above. For the digital identity card, for example, the issuing authority can delegate the rights to modify the address data to a local authority, but leave other data like the name or the date of birth immutable. Citizens would then be assured that the data has only been generated by (local or superior) authorities.[1]

Sanitizable signatures come with five security properties, described informally in [ACdMT05] and rigorously in [BFF+09]:

UNFORGEABILITY. Resembles the common unforgeability notion for regular signatures: besides the signer and the sanitizers no one should be able to produce signatures for new messages.

IMMUTABILITY. Confines the power of a malicious sanitizer, i.e., the sanitizer should not be able to change other parts of the message than the intended ones.

PRIVACY. Sanitization steps should remove *any* information about the original data of the sanitized parts. This is for instance important for the medical surveillance example, and usually holds in an information-theoretic sense.

[1] Note that this solves the modification problem but does not address the deniability issues discussed before. Still, for applications where the receiver is considered to be trustworthy, say, the police, deniability may be a minor issue. In addition, since our solution below is for example rather generic, it can potentially be combined with privacy-enhancing solutions in order to overcome the deniability problem.

ACCOUNTABILITY. In case of a dispute about the origin each party can contribute to settle the dispute. A malicious signer or sanitizer cannot frame the other party.

TRANSPARENCY. One cannot distinguish between signatures created by the signer or the sanitizer.

The aforementioned work of Brzuska et al. [BFF+09] defined these properties with game-based definitions and gave a construction based on the protocol in [ACdMT05], provably meeting these five requirements. The signature length, however, is quite large and the computational overhead grows with the number of admissible modifications. The construction also relies on specific number-theoretic assumptions.

Our results. We show that dropping the transparency requirement —thus allowing to distinguish genuine signatures of the signer from signatures produced by the sanitizer— yields significantly more efficient solutions: We present a construction allowing short signatures, signature generation time comparable to regular signatures and based on arbitrary (but secure) signature schemes.

Basically, the signer in our construction signs the fixed message parts m_{FIX} and the description of the admissible modifications ADM together with the sanitizer's public key pk_{san} to get a signature σ_{FIX}. In addition, the signer generates another signature σ_{FULL} for the entire message (including modifiable parts). Then the full signature is given by $\sigma = (\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM}, pk_{\text{san}})$.

To sanitize the message and replace (some of) the modifiable message parts the sanitizer changes the message m to m' accordingly and then creates the new signature σ' by signing m' with its signing key and replacing σ_{FULL} by the derived signature σ'_{FULL}. The entire signature for the sanitized message is given by $\sigma' = (\sigma_{\text{FIX}}, \sigma'_{\text{FULL}}, \text{ADM}, pk_{\text{san}})$.

We show that the construction above achieves unforgeability, immutability, accountability and privacy. It is clearly *not* transparent as one can easily distinguish under whose public key the second signature component verifies. As for the identity card example, transparency is usually neither provided by the solutions for "non-digital" identity cards, because the sticker is clearly visible given the card. Still, transparency may be a desirable security goal in some settings, say, if a recent change of the address entails discrimination. An example might be a landlord who is only willing to rent out to tenants which have not moved recently.

Our solution comes with several advantages over previous approaches, besides its generality and efficiency improvements. First, since we analyze the solution in terms of the security notions of [BFF+09] for sanitizable signatures, the solution really guarantees the desired goals, and these formally stated goals can be scrutinized. Also, our solution allows handy hierarchical extensions. That is, the sanitizer is allowed to change parts of a message, and can authorize a subordinate authority to modify some of these parts. To this end, the sanitizer issues certificates for public keys of local authorities such that they can make further modifications by replacing the second signature component and appending their public key together with the certificate to the signature.

2 Outline of the Construction

Our construction works as follows: Both the signer and the sanitizer each hold a key pair $(sk_{\text{sig}}, pk_{\text{sig}})$ and $(sk_{\text{san}}, pk_{\text{san}})$, respectively, of a secure signature scheme. The signature schemes used by the signer and the sanitizer can be distinct but we use the same scheme for sake of simplicity. To sign a message m and allowing modifications by the sanitizer with public key pk_{san}, the signer first picks a description ADM of the admissible message parts which are changeable by the sanitizer, and those parts m_{FIX} which are fixed. Then the signer computes the signature by signing the fixed part and the entire message (prepended with a bit to indicate the difference):

$$\sigma = (\sigma_{\text{FIX}}, \sigma_{\text{FULL}}) = (\mathsf{Sign}(sk_{\text{sig}}, (0, m_{\text{FIX}}, \text{ADM}, pk_{\text{san}})),\ \mathsf{Sign}(sk_{\text{sig}}, (1, m, pk_{\text{san}}, pk_{\text{sig}}))).$$

We assume that ADM (and possibly pk_{san}, if not linked to the signature somewhere else) become part of the signature. As an example, ADM might be of the form $(t, 6, 110000)$, indicating that a message consists of 6 blocks, each of bit length t and the sanitizer is allowed to change the first two blocks.

The sanitizer can now modify the message, yielding message m', and replace the signature part σ_{FULL} with a self-generated signature under pk_{san} (but leaving σ_{FIX} untouched):

$$\sigma' = (\sigma_{\text{FIX}}, \sigma'_{\text{FULL}}) = (\sigma_{\text{FIX}}, \mathsf{Sign}(sk_{\text{san}}, (1, m', pk_{\text{san}}, pk_{\text{sig}}))).$$

To verify a signature σ resp. σ' for a message m with respect to pk_{sig} the verifier first recovers the fixed part m_{FIX} by inspecting ADM. Then the verifier checks the validity of the signature part σ_{FIX} with respect to $(0, m_{\text{FIX}}, \text{ADM}, pk_{\text{san}})$, and then verifies that the second part of the signature either verifies under the signer's or the sanitizer's public key. If both properties hold then the verifier accepts.

Let us briefly revisit the security notions for sanitizable signatures [ACdMT05, BFF+09] and discuss if the scheme above achieves these notions. A formal approach follows in the next section.

UNFORGEABILITY. We need to argue that no one except for the signer and the designated sanitizer can create valid signatures for new messages. The unforgeability of the underlying signature scheme guarantees that one cannot forge signatures for the fixed part, including pk_{san}, and thus any forgery for the second part must necessarily be either for the sanitizer's public key or the signer's public key. But then the unforgeability of the sanitizer's and signer's signatures guarantee security for our sanitizable scheme. Note that prepending the bit 0 and 1 to the messages in the two signatures prevents "mix-and-match" attacks in which the adversary abuses the first signature component for the second part.

IMMUTABILITY. Guarantees that a malicious sanitizer cannot change inadmissible blocks. This follows from the unforgeability of the signer's scheme, protecting the fixed part of the message.

PRIVACY. Message parts which are replaced cannot be recovered, because the sanitizer removes those parts and signs the derived message from scratch. The information about the original data is hidden information-theoretically.

ACCOUNTABILITY. Neither party can claim that a message-signature pair originates from the other party, unless this party has really signed the corresponding message before. This again follows from the unforgeability of the underlying signature scheme. Note that, in practice, this may require some certification of the owner of the sanitizer's public key pk_{san}, or else the signer could create fake public keys on behalf of the sanitizer.

TRANSPARENCY. Does not hold. One can easily distinguish signatures generated by the signer from those produced by the sanitizer by inspecting the second signature part.

An interesting feature of the solution above is that the sanitizer itself can now act as a certificate authority and delegate rights further. To allow a subordinate sanitizer the sanitizer now acts as the signer and generates σ_{FULL} as $(\sigma^{\text{san}}_{\text{FIX}}, \sigma^{\text{san}}_{\text{FULL}})$ by dividing the message further into a part $m^{\text{san}}_{\text{FIX}}$ which the subordinate sanitizer should not be allowed to change, and into a variable part. The lack of transparency then again allows to decide upon the origin.

3 Technical Details of the Construction

We first present the formal structure of sanitizable signatures and then introduce our construction according to this structure. We next discuss the security notions in detail and finally show that our construction is secure according to these notions.

3.1 Sanitizable Signatures

The following definitions are taken from [BFF+09]. With our solution in mind, we simplify the presentation whenever possible. For example, our solution does not require an explicit Proof algorithm to identify the origin (signer or sanitizer), so we drop it from the formal descriptions.

Recall that our construction is based on a regular signature scheme $\mathcal{S} = (\mathsf{SKGen}, \mathsf{SSign}, \mathsf{SVf})$ which consists of three efficient algorithms where SKGen on input 1^n, the security parameter in unary, returns a key pair (sk_0, pk_0); algorithm SSign on input sk_0 and a message $m \in \{0,1\}^*$ returns a signature σ; and algorithm SVf for input pk_0, m, σ returns a decision bit d for accept ($d = 1$) or reject ($d = 0$). We assume completeness in the sense that any signature generated via SSign is also accepted by SVf. *Unforgeability under adaptive chosen message attacks* of regular signature schemes says that for any efficient algorithm $\mathcal{A}$ the probability that $\mathcal{A}$ with input pk_0 and access to a signing oracle $\mathsf{SSign}(sk_0, \cdot)$ for $(sk_0, pk_0) \leftarrow \mathsf{SKGen}(1^n)$ outputs a pair (m^*, σ^*) such that $\mathsf{SVf}(pk_0, m^*, \sigma^*) = 1$ and m^* has never been submitted to the signing oracle, is negligible.

A sanitizable signature scheme SanSig is now a tuple of efficient algorithms $(\mathsf{KGen}_{\text{sig}}, \mathsf{KGen}_{\text{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Judge})$ such that:

KEY GENERATION. The key generation algorithms for the signer and sanitizer, respec-

tively, allows both parties to generate key pairs (for security parameter n, given as input):

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

SIGNING. The signing algorithm of the signer takes the signer's secret key sk_{sig}, a message $m \in \{0,1\}^*$ the public key pk_{san} of the designated sanitizer and a description ADM (used to identify the fixed part m_{FIX} of m). It outputs a signature (or $\perp$, indicating an error):

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

We assume that ADM, pk_{san} are recoverable from any signature $\sigma \neq \perp$.

SANITIZING. The sanitizer's 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 first modifies the message m according to the modification instruction MOD and then computes a new signature σ' for the modified message m'. It outputs m' and σ' (or possibly $\perp$ in case of an error).

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

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

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

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

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

As usual we demand minimalistic functional properties of sanitizable signature schemes such that the verifier always accepts signatures generated by the honest signer or sanitizer, and that the judge decides correctly if the data has been formed correctly.

3.2 Construction

In order to describe our scheme formally we assume that ADM and MOD are (descriptions of) efficient deterministic algorithms such that MOD maps any message m to the modified message $m' = \text{MOD}(m)$, and $\text{ADM}(\text{MOD}) \in \{0,1\}$ indicates if the modification is admissible and matches ADM, i.e., $\text{ADM}(\text{MOD}) = 1$. For example, for messages $m = m[1] \ldots m[k]$ divided into blocks $m[i]$ of equal bit length t, ADM might contain t and the indices of the modifiable blocks, and MOD essentially consists of pairs $(j, m'[j])$ defining the new value for the j-th block.

We also let FIX_{ADM} be an efficient deterministic algorithm which is uniquely determined by ADM and which maps m to the immutable message part $m_{\text{FIX}} = \text{FIX}_{\text{ADM}}(m)$, e.g., for block-divided messages m_{FIX} is the concatenation of all blocks not appearing in ADM. To exclude trivial examples we demand that admissible modifications leave the fixed part of a message unchanged, i.e., $\text{FIX}_{\text{ADM}}(m) = \text{FIX}_{\text{ADM}}(\text{MOD}(m))$ for all $m \in \{0,1\}^*$, MOD with $\text{ADM}(\text{MOD}) = 1$. In addition, we also need that the fixed part must be maximal given ADM, i.e., $\text{FIX}_{\text{ADM}}(m') \neq \text{FIX}_{\text{ADM}}(m)$ for $m' \notin \{\text{MOD}(m) \mid \text{MOD with } \text{ADM}(\text{MOD}) = 1\}$ (else FIX_{ADM} mapping to the empty string would for example be a valid instantiation).

Construction 3.1 (Sanitizable Signature Scheme) *Let $\mathcal{S} = (\mathsf{SKGen}, \mathsf{SSign}, \mathsf{SVf})$ be a regular signature scheme. Define the sanitizable signature scheme $\mathsf{SanSig} = (\mathsf{KGen}_{sig}, \mathsf{KGen}_{san}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Judge})$ as follows:*

KEY GENERATION. *Algorithm KGen_{sig} generates on input 1^n a key pair $(pk_{sig}, sk_{sig}) \leftarrow \mathsf{SKGen}(1^n)$ of the underlying signature scheme, and algorithm KGen_{san} for input 1^n analogously returns a pair $(pk_{san}, sk_{san}) \leftarrow \mathsf{SKGen}(1^n)$.*

SIGNING. *Algorithm Sign on input $m \in \{0,1\}^*$, sk_{sig}, pk_{san}, ADM sets $m_{\text{FIX}} = \text{FIX}_{\text{ADM}}(m)$ for the algorithm FIX_{ADM} determined by ADM, and computes*

$$\sigma_{\text{FIX}} = \mathsf{SSign}(sk_{sig}, (0, m_{\text{FIX}}, \text{ADM}, pk_{san})), \sigma_{\text{FULL}} = \mathsf{SSign}(sk_{sig}, (1, m, pk_{san}, pk_{sig}))$$

using the underlying signing algorithm. It returns $\sigma = (\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM})$.

SANITIZING. *Algorithm Sanit on input a message m, information MOD, a signature $\sigma = (\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM})$, keys pk_{sig} and sk_{san} first recovers $m_{\text{FIX}} = \text{FIX}_{\text{ADM}}(m)$. It then checks that MOD is admissible according to ADM and that σ_{FIX} is a valid signature for message $(0, m_{\text{FIX}}, \text{ADM}, pk_{san})$ under key pk_{sig} (for pk_{san} included in sk_{san}). If not, it stops outputting $\perp$. Else, it derives the modified message $m' = \text{MOD}(m)$ and computes*

$$\sigma'_{\text{FULL}} = \mathsf{SSign}(sk_{san}, (1, m', pk_{san}, pk_{sig}))$$

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

VERIFICATION. *Algorithm Verify on input a message $m \in \{0,1\}^*$, a signature $\sigma = (\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM})$ and public keys pk_{sig} and pk_{san} first recovers $m_{\text{FIX}} = \text{FIX}_{\text{ADM}}(m)$. It then checks that $\mathsf{SVf}(pk_{sig}, (0, m_{\text{FIX}}, \text{ADM}, pk_{san}), \sigma_{\text{FIX}}) = 1$ accepts σ_{FIX} as a valid signature and that either $\mathsf{SVf}(pk_{sig}, (1, m, pk_{san}, pk_{sig}), \sigma_{\text{FULL}})$ or $\mathsf{SVf}(pk_{san}, (1, m, pk_{san}, pk_{sig}), \sigma_{\text{FULL}})$ verifies as $\texttt{true}$, too. If so, it outputs 1, declaring the entire signature as valid. Otherwise it returns 0, indicating an invalid signature.*

JUDGE. *The judge on input $m, \sigma, pk_{sig}, pk_{san}$ parses σ as $(\sigma_{\text{FIX}}, \sigma_{\text{FULL}}, \text{ADM})$ and outputs $\texttt{Sig}$ if $\mathsf{SVf}(pk_{sig}, (1, m, \text{ADM}, pk_{san}), \sigma_{\text{FULL}})$ validates as true, else if $\mathsf{SVf}(pk_{san}, (1, m, pk_{san}, pk_{sig}) = 1$ then it returns $\texttt{San}$. Note that one of these two verification must work, as Judge is only run on valid pairs (m, σ).*

Completeness of signatures generated by the signer and sanitizer follows easily from the completeness of the underlying signature scheme 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, because one can easily circumvent this problem by having each party also prepend a bit to the signature, indicating the origin (0 for signer and 1 for sanitizer). The judge can then also check that this bit matches its decision.

3.3 Security of Sanitizable Signatures

Here we recall the security notions for sanitizable signatures given by Brzuska et al. [BFF+09] (except for transparency which we do not define formally since our scheme does not achieve it). We note that Brzuska et al. [BFF+09] 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. However, since we drop the latter requirement we need to show privacy from scratch. In this version we omit the formal descriptions of the security properties for space reasons.

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 tupel (m^*, σ^*) such that $\mathsf{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk_{\text{san}}) = \texttt{true}$ without having the secret keys $sk_{\text{sig}}, sk_{\text{san}}$. This must hold even if one can see additional signatures for other 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.

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 ADM_i and keys $pk_{\text{san},i}$ of its choice, before eventually outputting a valid pair (m^*, σ^*) and pk^*_{san} 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 $m^* \notin \{\text{MOD}(m) \mid \text{MOD with } \text{ADM}_i(\text{MOD}) = 1\}$ for the value ADM_i in σ_i, i.e., that m^* and m_i differ in at least one inadmissible block. Again, giving the adversary the possibility to ask the signer about other sanitizer keys $pk_{\text{san},i}$ covers the case that the signer interacts with several sanitizers at the same time.

Accountability. Accountability says that the origin of a (sanitized) signature should be undeniable. There are 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 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 messages previously signed by the Sign oracle) such that the judge still outputs "$\texttt{Sig}$", i.e., that the signature has been created by the signer.

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 $(m_i, \text{MOD}_i, \sigma_i, pk_{\text{sig},i})$ receiving answers (m'_i, σ'_i). Adversary $\mathcal{A}_{\mathsf{Sign}}$ finally outputs a tuple $(pk^*_{\text{sig}}, m^*, \sigma^*)$ and is considered to succeed if Judge accuses the sanitizer for the new key-message pair pk^*_{sig}, m^* with a valid signature σ^*.

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.

Our approach is based on an indistinguishability notion[2] where an adversary can choose pairs (m_0, MOD_0), (m_1, MOD_1) of messages and modifications together with a description ADM and has access to a "left-or-right" box. This oracle either returns a sanitized signature for the left tuple ($b = 0$) or for the right tuple ($b = 1$). The task of the attacker is to predict the random bit b significantly better than by guessing. Here we need the additional constraint that for each call to the left-or-right box the resulting modified messages are identical for both tuples and the modifications both match ADM, else the task would be trivial. We write $(m_0, \text{MOD}_0, \text{ADM}) \equiv (m_1, \text{MOD}_1, \text{ADM})$ for this.

3.4 Security of Our Construction

Theorem 3.2 *The sanitizable signature scheme in Construction 3.1 provides unforgeability, immutability, privacy and accountability.*

Proof. We only need to show immutability, accountability and privacy, as the signer- and sanitizer-accountability together imply unforgeability [BFF+09].

Immutability. Assume towards contradiction that our construction is not immutable. We show that this contradicts the unforgeability of the underlying signer's signature scheme, i.e., we show that an adversary who successfully breaks immutability can be used to forge signatures under the signer's public key.

Let $\mathcal{A}$ be an efficient successful adversary against immutability. Adversary $\mathcal{A}$ impersonates the sanitizer and has access to a signing oracle $\mathsf{Sign}(\cdot, sk_{\text{sig}}, \cdot, \cdot)$. We show that if $\mathcal{A}$ is able to find $(m^*, \sigma^*, pk^*_{\text{san}})$ such that $\mathsf{Verify}(m^*, \sigma^*, pk_{\text{sig}}, pk^*_{\text{san}}) = \texttt{true}$ and for all queries to

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

the signing oracle we have $pk^*_{\text{san}} \neq pk_{\text{san},i}$ or $m^* \notin \{\text{MOD}(m_i) \mid \text{ADM}(\text{MOD}) = 1\}$, then the forgery immediately gives rise to a forgery against the underlying signature scheme.

The validity of the sanitizable signature σ^* in the adversary's forgery attempt contains a valid signature σ^*_{FIX} for $(0, m^*_{\text{FIX}}, \text{ADM}^*, pk^*_{\text{san}})$ under the signer's public key, it thus suffices to show that this tuple has not been input into the signing algorithm. First observe that since the signatures for the entire message start with a 1-bit, we only need to consider signatures created for tuples with 0-bits. Hence, if $(0, m^*_{\text{FIX}}, \text{ADM}^*, pk^*_{\text{san}}) = (0, m_{\text{FIX},i}, \text{ADM}_i, pk_{\text{san},i})$ for a query then $\text{ADM}_i = \text{ADM}^*$ and $\text{FIX}_{\text{ADM}}(m^*) = \text{FIX}_{\text{ADM}}(m_i)$, thus (by assumption about FIX_{ADM}) m^* must be a valid modification $\text{MOD}(m_i)$ of m_i. Therefore this forgery attempt cannot satisfy the requirement $pk^*_{\text{san}} \neq pk_{\text{san},i}$ or $m^* \notin \{\text{MOD}(m_i) \mid \text{ADM}(\text{MOD}) = 1\}$.

Note that the formal argument requires to build an adversary $\mathcal{B}$ against the underlying signature scheme with oracle access to a signing oracle of that scheme. Then one shows that $\mathcal{B}$ can simulate $\mathcal{A}$'s attack on the sanitizable scheme and, in particular, the signer oracle in the immutability attack. But this is straightforward for our scheme, given the signing oracle of the underlying signature scheme.

Sanitizer-accountability. We show that if the sanitizer can make the judge falsely accuse the signer, then the sanitizer can break the unforgeability of the underlying signer's signature scheme. Let $\mathcal{A}_{\mathsf{Sanit}}$ be an efficient and successful attacker. She has access to a signing oracle $\mathsf{Sign}(\cdot, sk_{\text{sig}}, \cdot, \cdot)$ and outputs a fresh, valid triple $(pk^*_{\text{san}}, m^*, \sigma^*)$, where $(pk^*_{\text{san}}, m^*) \neq (pk_{\text{san},i}, m_i)$ for all $(pk_{\text{san},i}, m_i, \text{ADM}_i)$-queries to the signing oracle.

The triple output by $\mathcal{A}_{\mathsf{Sanit}}$ is such that $\mathsf{Judge}(m^*, \sigma^*, pk_{\text{sig}}, pk^*_{\text{san}}) = \texttt{Sig}$. This means that Judge considers σ^*_{FULL} and notices that σ^*_{FULL} is a valid signer signature for the message $(1, m^*, pk^*_{\text{san}}, pk_{\text{sig}})$. But since $(pk^*_{\text{san}}, m^*) \neq (pk_{\text{san},i}, m_i)$ for all i and as all signatures for the fixed part are signatures over messages prepended with a 0-bit, it follows that $(1, m^*, pk^*_{\text{san}}, pk_{\text{sig}})$ has not been signed before. The formal argument (building an adversary $\mathcal{B}$ against the signature scheme, mounting a black-box simulation of $\mathcal{A}$) follows again straightforwardly.

Signer Accountability. We show that a successful attacker against signer accountability can be used to forge signatures of the sanitizer's signature scheme. Let $\mathcal{A}_{\mathsf{Sign}}$ be an efficient successful adversary. She is given access to a sanitizing oracle, respectively, the sanitizer's signing oracle $\mathsf{Sign}(\cdot, sk_{\text{san}}, \cdot, \cdot)$ and outputs a fresh, valid triple $(pk^*_{\text{sig}}, m^*, \sigma^*)$, where $(pk^*_{\text{sig}}, m^*) \neq (pk_{\text{sig,i}}, m_i)$ for all $(m_i, \text{MOD}_i, \sigma_i, pk_{\text{sig,i}}, sk_{\text{san}})$-queries to the sanitizing oracle.

The triple output by the adversary is such that $\mathsf{Judge}(m^*, \sigma^*, pk^*_{\text{sig}}, pk_{\text{san}}) = \texttt{San}$, i.e., Judge inspects σ^*_{FULL} and verifies that σ^*_{FULL} is a valid sanitizer signature for the message $(1, m^*, pk^*_{\text{san}}, pk_{\text{sig}})$. Since the sanitizer only signs messages beginning with 1 and $(pk^*_{\text{sig}}, m^*) \neq (pk_{\text{sig,i}}, m_i)$ for all queries, it follows that the sanitizer has not input this message into its signature algorithm before. The forgery thus comprises a forgery for the basic signature scheme,

Privacy. Privacy is guaranteed information-theoretically: Since the left-or-right oracle only receives message pairs and modifications mapping to the same outcome, and the sanitizer signs this derived message from scratch, the output distribution is identical for both values of the bit b in the left-or-right oracle. □

3.5 Variations and Extensions

Our generic construction easily allows variations and extensions like hierarchical sanitizing. The sanitizer can delegate some of his rights to a subordinate sanitizer as follows. Let

$$(\sigma_{\text{FIX}}, \sigma_{\text{FULL}}) = (\mathsf{Sign}(sk_{\text{sig}}, (0, m_{\text{FIX}}, \text{ADM}, pk_{\text{san}})),\ \mathsf{Sign}(sk_{\text{sig}}, (1, m, pk_{\text{san}}, pk_{\text{sig}})))$$

be a signer's signature for the message m. It is clear that the sanitizer can only delegate rights concerning the admissible blocks of the message. He thus determines a "subset" $\text{ADM}_{\text{sub}} \subseteq \text{ADM}$ (with the meaning that $\text{ADM}(\text{MOD}) = 1$ whenever $\text{ADM}_{\text{sub}}(\text{MOD}) = 1$) that the subordinate sanitizer is allowed to modify. Let m' be the sanitizer's modification of the message m, and $\text{FIX}_{\text{ADM}_{\text{sub}}}$ map m' to the concatenation $m'_{\text{FIX},sub}$ of the message parts which are immutable for the subordinate sanitizer. Let pk_{SubSan} be the subordinate sanitizer's public key.

To delegate the rights the sanitizer now signs the messages

$$(2, m'_{\text{FIX,sub}}, \text{ADM}_{\text{sub}}, pk_{\text{sig}}, pk_{\text{SubSan}}) \text{ and } (3, m', pk_{\text{SubSan}}, pk_{\text{sig}}).$$

to obtain $\sigma^{\text{san}}_{\text{FIX}}$ and $\sigma^{\text{san}}_{\text{FULL}}$. The signature issued by the sanitizer consists of

$$(\sigma_{\text{FIX}}, \text{ADM}, \sigma^{\text{san}}_{\text{FIX}}, \sigma^{\text{san}}_{\text{FULL}}, \text{ADM}_{\text{sub}})$$

and possibly all the public keys. For sanitizing m' to m'', the subordinate sanitizer algorithm $\mathsf{SubSanit}$ leaves $(\sigma_{\text{FIX}}, \text{ADM}, \sigma^{\text{san}}_{\text{FIX}}, \text{ADM}_{\text{sub}})$ unchanged and creates a new signature $\sigma^{\text{san}}_{\text{FULL}}{}' = \mathsf{SSign}(pk_{\text{SubSan}}, (3, m'', pk_{\text{SubSan}}, pk_{\text{sig}}))$. As the final signature it outputs

$$(\sigma_{\text{FIX}}, \text{ADM}, \sigma^{\text{san}}_{\text{FIX}}, \sigma^{\text{san}}_{\text{FULL}}{}', \text{ADM}_{\text{sub}}).$$

Further hierarchical levels of sanitizers can be added accordingly.

Concerning flat hierarchies, in some settings it may be desirable to involve several sanitizer, say, a setting with personnel in a hospital. The extension of our scheme to such a setting is straightforward. For each message the authorized sanitizer set is chosen, and σ_{FIX} is a signer's signature over the message $(0, m_{\text{FIX}}, \text{ADM}, pk_{\text{san},1}, \ldots, pk_{\text{san},k})$, where $pk_{\text{san},1}, \ldots, pk_{\text{san},k}$ are the authorized sanitizers' public keys. In addition, let σ_{FULL} be a signer's or sanitizer's signature over the message $(1, m, pk_{\text{san},1}, \ldots, pk_{\text{san},k}, pk_{\text{sig}})$. For verifying the validity of a signature, one checks that σ_{FIX} is a valid signer signature over $(0, m_{\text{FIX}}, \text{ADM}, pk_{\text{san},1}, \ldots, pk_{\text{san},k})$ and that σ_{FULL} verifies for the message $(1, m, pk_{\text{san},1}, \ldots, pk_{\text{san},k}, pk_{\text{sig}})$ under pk_{sig} or under one of the authorized sanitizers' keys $pk_{\text{san},1}, \ldots, pk_{\text{san},k}$ (this key may be determined as part of the signature).

The construction described above produces signatures which are linear in the number of sanitizers. This shall be avoided in settings involving a huge number of sanitizers. In this case, instead of signing $(0, m_{\text{FIX}}, \text{ADM}, pk_{\text{san},1}, \ldots, pk_{\text{san},k})$, one signs $(0, m_{\text{FIX}}, \text{ADM}, pk_{CA})$, where pk_{CA} is the public verification key of a certificate authority, which provides certificates for each sanitizer key $pk_{\text{san},k}$. The sanitizer then attaches his $pk_{\text{san},k}$ as well as its certificate to the signature. Certificates are endowed with an expiration date so that keys are changed regularly. Therefore, it is necessary that the fixed part m_{FIX} of the message contains some information about the signing date, which is the case for identity cards.

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. In *ESORICS*, volume 3679 of *Lecture Notes in Computer Science*, pages 159–177. Springer, 2005.

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

[BKMN08] Jens Bender, Dennis Kügler, Marian Margraf, and Ingo Naumann. Sicherheitsmechanismen für kontaktlose Chips im deutschen elektronischen Personalausweis. In *DuD — Datenschutz und Datensicherheit*, volume 3, pages 164–177. Vieweg, 2008.

[Bun08] Bundesministerium des Innern. Grobkonzept zur Einführung des elektronischen Personalausweises. (Version 2.0), July 2008.

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

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