On Extended Sanitizable Signature Schemes*

Sébastien Canard[1] and Amandine Jambert[1,2]

[1] Orange Labs, 42 rue des Coutures, BP6243, 14066 Caen Cedex, France
[2] IMB, Université Bordeaux 1, 351 cours de la Libération, 33405 Talence, France

Abstract. Sanitizable signature schemes allow a semi-trusted entity to modify some specific portions of a signed message while keeping a valid signature of the original off-line signer. In this paper, we give a new secure sanitizable signature scheme which is, to the best of our knowledge, the most efficient construction with such a high level of security. We also enhance the Brzuska *et al.* model on sanitizable signature schemes by adding new features. We thus model the way to limit the set of possible modifications on a single block, the way to force the same modifications on different admissible blocks, and the way to limit both the number of modifications of admissible blocks and the number of versions of a signed message. We finally present two cryptanalysis on proposals for two of these features due to Klonowski and Lauks at ICISC 2006 and propose some new practical constructions for two of them.
Keywords. Sanitizable signature, chameleon hash, accumulator scheme.

1 Introduction

Since the appearance of public key cryptography, signature schemes have been one of the most widely studied cryptographic tool. Among some others, one of the main security properties a signature scheme should verify is the integrity of the message. However, in some cases, such as medical applications, secure routing or content protection [1, 5], it may be necessary for a designated semi-trusted entity to delete or modify some parts of the signed message.

In this paper, we focus on sanitizable signature schemes (introduced in [1] and later formalized in [2]) which permits a signer to produce a signature on a document, which can be further modified, in a limited and controlled fashion, by a designated semi-trusted "sanitizer", with no interaction with the original signer. Moreover, the signature on the resulting message should be verifiable as a signature from the original signer. On the other hand, the sanitizer should be able to modify only the sanitizable parts of the message, that is, the parts that have been stated as modifiable/admissible by the signer.

1.1 Related Work

The first sanitizable signature scheme [1] makes use of chameleon hash functions [8] and this is also the case for most of existing ones. The first problem with

* This work has been financially supported by the French Agence Nationale de la Recherche and the TES Cluster under the PACE project.

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

this construction is the possibility to obtain some new sanitized messages from two different ones, without the secret key of the sanitizer (i.e. it is forgeable). Moreover, a judge is unable to decide whether a signature has been sanitized or not. Thus, according to [2], the scheme is not accountable.

Canard *et al.* [5] fix both problems, in the context of trapdoor sanitizable signature, by adding an extra modifiable block corresponding to the whole message. The original message is used as a unique identifier to obtain accountability. However, as the original message is obviously recognizable from a sanitized one, this scheme is not transparent [2].

In [2], Brzuska *et al.* propose the first secure scheme (i.e. immutable, transparent and accountable). They propose to add a tag (verifiably and pseudo-randomly generated by the signer and randomly by the sanitizer) to each modifiable block. Thus, the signer can prove which one she constructed. Unforgeability (and thus accountability) is reached thanks to the computation of a new tag per message. This implies to compute a collision for each modifiable block, even if this block has not been modified. Thus, this solution lacks of efficiency.

Yuen *et al.*[10] also give a solution in the standard model but without accountability.

At ICISC'06, Klonowski and Lauks propose [7] several extensions of the sanitization signature paradigm: force the sanitizer to construct less than l versions of a message, modify at maximum k sanitizable blocks or limit the values available for some blocks. However neither security model nor proofs are given.

1.2 Our Contribution

In this paper, we provide several contributions to sanitizable signature schemes. We first extend in Section 4 the Brzuska *et al.* [2] model (see Section 2) by taking into account the way to (i) limit block modifications in a set, (ii) secretly force the same modifications on different admissible blocks (which can be different at the beginning), (iii) limit the number of admissible blocks modifiable and (iv) limit the number of versions of a signed message. We also give in Section 3 a new sanitizable signature scheme without additional features which is, to the best of our knowledge, the most efficient and secure construction. After that, we show in Section 5 a cryptanalysis on two proposals for additional features (ii) and (iii) due to Klonowski and Lauks [7]. Finally, we present in Section 6 practical constructions for the extensions (iii) and (iv) and show how the idea from Klonowski and Lauks for (i) can be made secure.

2 Initial Model for Sanitizable Signatures

In the following, the size of the message (in bits) is denoted ℓ and a message is divided (by the signer) into t blocks. The variable ADM includes, for each block m_i, $i \in [1,t]$, the length ℓ_i of the corresponding i-th block (thus $\ell = \sum_{i=1}^{t} \ell_i$) and a subset of $[1,t]$ corresponding to the ranks of the blocks modifiables by the sanitizer (i.e. admissible). The variable MOD is a set of elements of the

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

form (i, m'_i). A value $(i, \mathsf{m}'_i) \in \mathsf{MOD}$ if and only if the i-th block is modified into m'_i during the sanitization. By misuse of notation, we denote $i \in \mathsf{MOD}$ if $\exists \mathsf{m}'_i / (i, m'_i) \in \mathsf{MOD}$. We say that MOD *matches* ADM if $\forall i \in \mathsf{MOD}, i \in \mathsf{ADM}$.

2.1 Procedures and Correctness

A *sanitizable signature scheme* $\mathcal{SS}$ is composed of the following algorithms (each of them may output an error $\perp$), where λ is a security parameter.

- SETUP takes as input 1^λ and outputs the parameters param of the system. In the following, we consider that λ is included into param.
- SIGKEYGEN (resp. SANKEYGEN) on input param outputs the key pair $(\mathsf{pk}_{sig}, \mathsf{sk}_{sig})$ for the signer (resp $(\mathsf{pk}_{san}, \mathsf{sk}_{san})$ for the sanitizer).
- SIGN takes as input a message m of length ℓ divided into t blocks, the secret key sk_{sig}, the public key pk_{san} and the variable ADM. It outputs a sanitizable signature σ on the message m. In the following, ADM is included into σ.
- SANITIZE takes as input a message m, a sanitizable signature σ, the public key pk_{sig}, the secret key sk_{san} and the modifications MOD that the sanitizer wants to do on m. It outputs a new signature σ' and message m'.
- VERIFY permits to verify a signature σ on a message m with the public keys pk_{sig} and pk_{san}. It outputs true if the signature is correct and false otherwise.
- PROOF takes as input a signature σ on a given message m, the secret key sk_{sig}, the public key pk_{san} and the set of message-signature pairs she has produced $(\mathsf{m}_i, \sigma_i)_{i=1,2,\cdots,q}$. It outputs a proof π.
- JUDGE is a public algorithm which aims at deciding who has produced a given signature. It takes as input (m, σ), a proof π from PROOF and the public keys pk_{sig} and pk_{san} and outputs signer or $\mathsf{sanitizer}$.

First, a sanitizable signature scheme needs to verify some correctness properties:

- **Signing correctness** says that a signature from SIGN with the secret key from SIGKEYGEN is accepted with an overwhelming probability by VERIFY.
- **Sanitizing correctness** says that a signature from SANITIZE from a valid signature with the secret key from SANKEYGEN is accepted with an overwhelming probability by VERIFY.
- **Proof correctness** says that for any sanitized message, the signer is able to output a proof π, using PROOF, such that JUDGE outputs $\mathsf{sanitizer}$.

2.2 Security Requirements

According to Brzuska *et al.*, a sanitizable signature scheme is secure if it verifies the following security properties. Formal experiments are given in the table below.

- **Immutability.** It is not possible for the sanitizer to modify non admissible blocks of a signed message. In the corresponding experiment, the adversary impersonates the sanitizer.

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

– **Transparency.** Only the signer and the sanitizer are able to distinguish an original signature from a sanitized one. During this experiment, the adversary is given access to a SIGN/SANIT oracle which on input a bit b outputs either a sanitized signature if $b = 0$ (output by SANITIZE) or a signed message if $b = 1$ (output by SIGN).
– **Accountability.** In case of an argument about the origin of a signature and a message, the judge is able to correctly settle it.

Immutability: $Succ^{SS}_{imm} = Pr[1 \longleftarrow \text{EXP}^{SS}_{imm}]$ where EXP^{SS}_{imm} is as follows – $(\mathsf{pk}_{sig}, \mathsf{sk}_{sig}) \longleftarrow \text{SIGKEYGEN}(1^\lambda)$ – $(\mathsf{pk}_{san}{}^*, \mathsf{m}^*, \sigma^*) \longleftarrow \mathcal{A}^{\text{SIGN}(\cdot,\mathsf{sk}_{sig},\cdot,\cdot),\text{PROOF}(\mathsf{sk}_{sig},\cdot,\cdot,\cdot)}(\mathsf{pk}_{sig})$ – Let $(\mathsf{m}_i, \mathsf{ADM}_i, \mathsf{pk}_{san,i})$ and σ_i for $i \in [1, q]$ be the queries related to oracle SIGN – return 1 if $\text{VERIFY}(\mathsf{m}^*, \sigma^*, \mathsf{pk}_{sig}, \mathsf{pk}_{san}{}^*) = \mathsf{true}$ and for all $i = 1, 2, \cdots, q$ we have – $\mathsf{pk}_{san}{}^* \neq \mathsf{pk}_{san,i}$ or – $\exists j_i \notin \mathsf{ADM}_i$ such that $\mathsf{m}^*[j_i] \neq \mathsf{m}_i[j_i]$
Transparency: $Adv^{SS}_{trans} = Pr[1 \longleftarrow \text{EXP}^{SS}_{trans}] - 1/2$ where EXP^{SS}_{trans} is as follows – $(\mathsf{pk}_{sig}, \mathsf{sk}_{sig}) \longleftarrow \text{SIGKEYGEN}(1^\lambda)$ – $(\mathsf{pk}_{san}, \mathsf{sk}_{san}) \longleftarrow \text{SANKEYGEN}(1^\lambda)$ – $b \longleftarrow \{0, 1\}$ – $b' \longleftarrow \mathcal{A}^{\text{SIGN}(\cdot,\mathsf{sk}_{sig},\cdot,\cdot),\text{SANIT}(\cdot,\cdot,\cdot,\mathsf{sk}_{san}),\text{PROOF}(\mathsf{sk}_{sig},\cdot,\cdot,\cdot),\text{SIGN/SANIT}(\cdot,\cdot,\cdot,\mathsf{sk}_{sig},\mathsf{sk}_{san},b)}(\mathsf{pk}_{sig}, \mathsf{pk}_{san})$ – return 1 if $b' = b$
Sanitizer Accountability: $Succ^{SS}_{san-acc} = Pr[1 \longleftarrow \text{EXP}^{SS}_{san-acc}]$ where $\text{EXP}^{SS}_{san-acc}$ is as follows – $(\mathsf{pk}_{sig}, \mathsf{sk}_{sig}) \longleftarrow \text{SIGKEYGEN}(1^\lambda)$ – $(\mathsf{pk}_{san}{}^*, m^*, \sigma^*) \longleftarrow \mathcal{A}^{\text{SIGN}(\cdot,\mathsf{sk}_{sig},\cdot,\cdot),\text{PROOF}(\mathsf{sk}_{sig},\cdot,\cdot,\cdot)}(\mathsf{pk}_{sig})$ – Let $(\mathsf{m}_i, \mathsf{ADM}_i, \mathsf{pk}_{san,i})$ and σ_i for $i \in [1, q]$ be the queries related to oracle SIGN – $\pi \longleftarrow \text{PROOF}(sk_{sig}, m^*, \sigma^*, \mathsf{m}_1, \sigma_1, \cdots, \mathsf{m}_q, \sigma_q, pk_{san}{}^*)$ – return 1 if $\text{VERIFY}(\mathsf{m}^*, \sigma^*, \mathsf{pk}_{sig}, \mathsf{pk}_{san}{}^*) = \mathsf{true}$ and – $(\mathsf{pk}_{san}{}^*, \mathsf{m}^*) \neq (\mathsf{pk}_{san,i}, \mathsf{m}_i)$ for all $i = 1, \cdots, q$ and – $\text{JUDGE}(\mathsf{m}^*, \sigma^*, pk_{sig}, pk_{san}{}^*, \pi) = \mathsf{signer}$
Signer Accountability: $Succ^{SS}_{sig-acc} = Pr[1 \longleftarrow \text{EXP}^{SS}_{sig-acc}]$ where $\text{EXP}^{SS}_{sig-acc}$ is as follows – $(\mathsf{pk}_{san}, \mathsf{sk}_{san}) \longleftarrow \text{SANKEYGEN}(1^\lambda)$ – $(\mathsf{pk}_{sig}{}^*, \pi^*, m^*, \sigma^*) \longleftarrow \mathcal{A}^{\text{SANIT}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{san})}(\mathsf{pk}_{san})$ – Let $(\mathsf{m}'_i, \sigma'_i)$ for $i = 1, 2, \cdots, q$ be the answers from oracle SANIT. – return 1 if $\text{VERIFY}(\mathsf{m}^*, \sigma^*, \mathsf{pk}_{sig}{}^*, \mathsf{pk}_{san}) = \mathsf{true}$ and – $(\mathsf{pk}^*_{sig}, \mathsf{m}^*) \neq (\mathsf{pk}_{sig,i}, \mathsf{m}'_i)$ for all $i = 1, \cdots, q$ and – $\text{JUDGE}(\mathsf{m}^*, \sigma^*, pk^*_{sig}, pk_{san}, \pi^*) = \mathsf{sanitizer}$

2.3 Useful Tools

Signature Schemes. We need a *signature scheme* $\mathcal{S}$ = (KEYGEN, SIGN, VERIFY) using a security parameter λ. The secret key is denoted ssk and the corresponding public verification key spk. The verification algorithm outputs true if the signature is correct and false if not. The used signature scheme needs to be existentially unforgeable against chosen message attacks (EU-CMA), that is $Succ^{\mathcal{S}}_{EU-CMA}$ is negligible in the security parameter [6].

Chameleon Hash Schemes. We will use a *chameleon hash scheme* $\mathcal{CH}$ = (SETUP, PROCEED, FORGE) using a security parameter λ. SETUP permits the generation of the key pairs $(\mathsf{chpk}, \mathsf{chsk})$ on input 1^λ. PROCEED takes as input the chameleon hash public key chpk, a message m and a random r and outputs the hash value h of the message m. FORGE, on input the chameleon hash secret key chsk, the message m, the random r, the hash value h and a new message m', outputs a new random r' such that $\mathsf{h} = \text{PROCEED}(\mathsf{chpk}, \mathsf{m}', \mathsf{r}')$.

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

A chameleon hash function is said strong (resp. weak) secure if it is both uniform (the distribution of the output of FORGE are indistinguishable from a random [8, 5]) and strong (resp. weak) collision resistant (it is impossible to find a collision $(\mathsf{m}', \mathsf{r}')$ on $\mathsf{h} = \text{PROCEED}(\mathsf{chpk}, \mathsf{m}, \mathsf{r})$, only having access to h, m, r, chpk and an oracle FORGE (resp. to h, m, r, chpk)). We note $Succ^{\mathcal{CH}}_{SCollRes}$ (resp. $Succ^{\mathcal{CH}}_{WCollRes}$) the success of an adversary against strong (resp. weak) collision and $Adv^{\mathcal{CH}}_{Uni}$ the advantage of an adversary against uniformity.

Pseudorandom Generators. We use in the following a pseudorandom generator PRG mapping λ-bits to 2λ-bits and a pseudorandom function PRF mapping λ-bits to λ-bits. We note $Adv^{PRG}_{Pseudorand}$ and $Adv^{PRF}_{Pseudorand}$ the advantages of an adversary against pseudo-randomness and Adv^{PRG}_{OneWay} the one-wayness of PRG.

Accumulator Schemes. An accumulator scheme [4, 9, 3] ACC permits to accumulate a large set of objects in a single short value, called the accumulator and denoted Acc. Such scheme provides evidence that a given object belongs to the accumulator by producing a witness w related to Acc and x by the relation $\mathsf{Acc} = \text{ACC}(x, w)$. We denote $x \in \mathsf{Acc}$, or $(x, w) \in \mathsf{Acc}$, if x is accumulated in Acc with the witness w. If someone reveals a value x together with a witness w, she proves that the value x is truly accumulated in the accumulator Acc iff $\text{ACC}(x, w) = \mathsf{Acc}$. Such scheme is divided into several procedures including the parameter generation which initializes the parameters, the accumulation phase to accumulate values in a new accumulator and the witnesses computation phase. Existing constructions provide the main security property of an accumulator scheme, named the collision resistant one, which says [9] that this is infeasible for an adversary, on input an accumulator Acc, to output a value and a witness that this value is accumulated in Acc, while this is not the case. In the following, we say that ACC is secure if $Succ^{\text{ACC}}_{CR}$ is negligible in the security parameter.

3 A New Construction in the Initial Model

3.1 High Level Description

Both [5] and [2] are based on Ateniese *et al.* [1], which works as follows. Let $\mathsf{m} = \mathsf{m}_1 \| \mathsf{m}_2 \| \cdots \| \mathsf{m}_t$ be the message to sign and ID_{m} a random unique identifier.

- The SIGN procedure consists in executing, for each admissible block m_i, $\mathcal{CH}$.PROCEED, with a random r_i, to obtain h_i. Then, the signer computes a modified version $\tilde{\mathsf{m}}_i$ of each block as either h_i if $i \in \mathsf{ADM}$ or $\mathsf{m}_i \| i$ otherwise. Finally, she executes the signature algorithm on the message $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1 \| \tilde{\mathsf{m}}_2 \| \cdots \| \tilde{\mathsf{m}}_t$, as $s = \mathcal{S}.\text{SIGN}(\mathsf{ssk}, ID_m || t || \mathsf{pk}_{san} || \tilde{\mathsf{m}})$. The final sanitizable signature σ on the message m is $(s, \mathcal{R}, \mathsf{ADM})$ where $\mathcal{R} = \{r_i : i \in \mathsf{ADM}\}$.
- The SANITIZE step from a message $\mathsf{m} = \mathsf{m}_1 \| \mathsf{m}_2 \| \cdots \| \mathsf{m}_t$ to a message $\mathsf{m}' = \mathsf{m}'_1 \| \mathsf{m}'_2 \| \cdots \| \mathsf{m}'_t$ consists in using $\mathcal{CH}$.FORGE to obtain the new r'_i for all $i \in \mathsf{ADM}$, so that the value h_i (and thus the signature s) is unchanged after the modification of the block message (m_i to m'_i).

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

This solution has two main problems. First, it is not accountable since a judge can not decide whether a signature has been sanitized or not. We propose to use a pseudorandom number TAG, instead of ID_{m}, which is linked to the version of the message and generated using both a PRG and a PRF [2]. Thus, the signer can prove that she has correctly computed the tag, while being transparent.

Second, it is forgeable since it is possible to obtain a new sanitization from two versions of the same message, without chsk. Let $\mathsf{m} = \mathsf{m}_1\|\mathsf{m}_2\|\mathsf{m}_3\|\mathsf{m}_4$, m_2 and m_4 are sanitized into $\mathsf{m}' = \mathsf{m}_1\|\mathsf{m}'_2\|\mathsf{m}_3\|\mathsf{m}'_4$. We obtain $(s, \{\mathsf{r}'_2, \mathsf{r}'_4\}, \mathsf{TAG})$ from $(s, \{\mathsf{r}_2, \mathsf{r}_4\}, \mathsf{TAG})$. Then everyone can obtain $(s, \{\mathsf{r}'_2, \mathsf{r}_4\}, \mathsf{ADM})$ on $\mathsf{m}_1\|\mathsf{m}'_2\|\mathsf{m}_3\|\mathsf{m}_4$. We add a final admissible block corresponding to the whole message [5]. As this block is updated at each version, the attack does not work any more.

3.2 Definition of Procedures

More formally, our sanitizable signature scheme works as follows.

- SigKeyGen. This step consists in executing $\mathcal{S}$.KeyGen to obtain $(\mathsf{ssk}, \mathsf{spk})$ and in choosing randomly a secret key κ in $\{0,1\}^\lambda$ for the PRF.
- SanKeyGen. This algorithm executes $\mathcal{CH}$.Setup to obtain $(\mathsf{sk}_{san}, \mathsf{pk}_{san})$.
- Sign. First, the signer generates the variable ADM as defined in the model. Let u be the number of modifiable parts in m. During this step, the signer generates the tag TAG by computing $x = \mathrm{PRF}(\kappa, \mathsf{Nonce})$ where $\mathsf{Nonce} \in \{0,1\}^\lambda$, and $\mathsf{TAG} = \mathrm{PRG}(x)$. In order to compute the chameleon hash function, she randomly chooses $\mathsf{r}_1, \cdots, \mathsf{r}_u, \mathsf{r}_c$ in $\{0,1\}^\lambda$. Then, for each admissible block, she executes what we call the (public) "reconstruction procedure", which takes as input the message m, TAG, the r_i's and the public key pk_{san}. It is divided into several steps.
  1. Compute the values $\tilde{\mathsf{m}}_i$ for each block:
  $$\forall i, \tilde{\mathsf{m}}_i = \begin{cases} \mathsf{h}_i = \mathcal{CH}.\textsc{Proceed}(\mathsf{pk}_{san}, \mathsf{m}_i||i, \mathsf{r}_i) & \text{if } \mathsf{m}_i \in \mathsf{ADM} \\ \mathsf{m}_i||i & \text{else} \end{cases}$$
  2. Compute the final block : $\mathsf{h}_c = \mathcal{CH}.\textsc{Proceed}(\mathsf{pk}_{san}, \mathsf{TAG}||\mathsf{m}, \mathsf{r}_c)$.

  After that, the signer signs the message $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1||\cdots||\tilde{\mathsf{m}}_t||\mathsf{h}_c||\mathsf{pk}_{san}$ as $s = \mathcal{S}.\textsc{Sign}(\mathsf{sk}_{sig}, \tilde{\mathsf{m}})$. She finally obtains the sanitizable signature $\sigma = (s, \mathsf{TAG}, \mathsf{Nonce}, \mathcal{R}, \mathsf{ADM})$ with $\mathcal{R} = \{\mathsf{r}_1, \cdots, \mathsf{r}_u, \mathsf{r}_c\}$. The signature and the message are added to the signer's database DB.
- Sanitize. The sanitizer uses the reconstruction procedure to obtain the h_i's and h_c. For all $i \in \mathsf{MOD}$, she finds a collision on h_i, using sk_{san}. She computes $\forall j \in \mathsf{MOD}, \mathsf{r}'_j = \mathcal{CH}.\textsc{Forge}(\mathsf{sk}_{san}, \mathsf{m}_j||j, \mathsf{m}'_j||j, \mathsf{h}_j)$ and $\mathsf{r}'_c = \mathcal{CH}.\textsc{Forge}(\mathsf{sk}_{san}, \mathsf{TAG}||\mathsf{m}, \mathsf{TAG}'||\mathsf{m}', \mathsf{h}_c)$, where Nonce' and TAG' are random values. The sanitized signature is $\sigma' = (s, \mathsf{TAG}', \mathsf{Nonce}', \mathcal{R}', \mathsf{ADM})$ where the set $\mathcal{R}' = \{\mathsf{r}'_1, \cdots, \mathsf{r}'_u, \mathsf{r}'_c\}$ (with $\mathsf{r}'_j = \mathsf{r}_j$ if $j \notin \mathsf{MOD}$).
- Verify. The verifier executes the reconstruction procedure as described above to obtain $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1||\cdots||\tilde{\mathsf{m}}_t||\mathsf{h}_c||\mathsf{pk}_{san}$. She finally returns the output of $\mathcal{S}.\textsc{Verify}(\mathsf{pk}_{sig}, s, \tilde{\mathsf{m}})$.
- Proof. The signer searches in DB an integer $i \in [1, q]$ such that

$$\mathcal{CH}.\textsc{Proceed}(\mathsf{pk}_{san}, \mathsf{TAG}||\mathsf{m}, \mathsf{r}_c) = \mathcal{CH}.\textsc{Proceed}(\mathsf{pk}_{san}, \mathsf{TAG}_i||\mathsf{m}_i, \mathsf{r}_{c_i}) \quad (1)$$

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

with $\mathsf{TAG}_i = \mathrm{PRG}(x_i)$ for $x_i = \mathrm{PRF}(\kappa, \mathsf{Nonce}_i)$ and $\mathsf{m} \neq \mathsf{m}_i$. If it exists, it outputs $\pi = (\mathsf{pk}_{sig}, \mathsf{TAG}_i, \mathsf{m}_i, \mathsf{r}_{c_i}, x_i)$ else, it outputs $\perp$.
- JUDGE. If $\pi = \perp$, then it returns signer. Else, $\pi = \{\mathsf{pk}_{sig}, \mathsf{TAG}_i, \mathsf{m}_i, \mathsf{r}_{c_i}, x_i\}$ and the algorithm checks if Equation (1) holds, with $\mathsf{m} \neq \mathsf{m}_i$ and $\mathsf{TAG}_i = PRG(x_i)$. If so it outputs sanitizer. If not, it outputs signer.

3.3 Security Considerations

Theorem 1. *Our scheme is secure if the signature scheme is EU-CMA, PRG and PRF are pseudo-random and $\mathcal{CH}$ is strong secure.*

Proof. We prove that our scheme is immutable, transparent and accountable.

- The **Immutability** is reached thanks to the fact that non-admissible blocks are directly signed with the EU-CMA signature scheme $\mathcal{S}$. More formally let $\mathcal{A}^{\mathcal{SS}}_{Imm}$ be an adversary against our scheme which, at the end of the experiment, outputs a message m^* and a public key pk^*_{san}.
  - It exists $\mathsf{m}^*_j \neq \mathsf{m}_{i_j}$ for some $j \notin \mathsf{ADM}$. We can use $\mathcal{A}^{\mathcal{SS}}_{Imm}$ to break the EU-CMA property of $\mathcal{S}$. Each time $\mathcal{A}^{\mathcal{SS}}_{Imm}$ queries a sanitizable signature from the signer, we use the signing oracle of $\mathcal{S}$. At the end, $\mathcal{A}^{\mathcal{SS}}_{Imm}$ outputs a valid new pair m^*, σ^*. As it exists $\mathsf{m}^*_j \neq \mathsf{m}_{i_j}$ for some $j \notin \mathsf{ADM}$, the underlying signed message $\tilde{\mathsf{m}}^*$ and the corresponding signature s^* give us a forge on the signature scheme $\mathcal{S}$.
  - $\mathsf{pk}^*_{san} \neq \mathsf{pk}_{san_i}$ on all requests. We first recall that the message signed by the signer is $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1 || \cdots || \tilde{\mathsf{m}}_t || \mathsf{h}_c || \mathsf{pk}_{san}$. By assumption, the underlying signed message $\tilde{\mathsf{m}}^*$ is different from all queried $\tilde{\mathsf{m}}_i$ on, at least, its last part corresponding to pk^*_{san}. All $\tilde{\mathsf{m}}_i$'s are signed using the signing oracle while the output signed message is a forge. We have thus broken the existential unforgeability of $\mathcal{S}$.

  As a consequence the probability of success of an adversary against the immutability of our scheme is $Succ^{\mathcal{SS}}_{Imm} \leq Succ^{\mathcal{S}}_{EU-CMA}$.
- The **Transparency** is satisfied since the outputs of the signature and sanitization algorithms are similar, except in the construction of TAG and r_i.
  - Let us first focus on the r_i's. In this case, the transparency property results in the distributional property of the chameleon hash function. During a SIGN procedure, the r_i for the $\mathcal{CH}$.PROCEED algorithm are chosen at random while during the SANITIZE algorithm, the r_i's corresponds to the outputs of $\mathcal{CH}$.FORGE. Thus, the probability of success of the adversary in this case is the same as against the uniformity property of the chameleon hash function, $Adv^{\mathcal{CH}}_{Uni}$.
  - Regarding TAG, the signer chooses at each new signature a new pseudo-random value Nonce and uses it to compute the value TAG thanks to PRF and PRG. Thus, TAG is indistinguishable from a random value, under the pseudorandomness of functions PRF and PRG.

  As a conclusion, the advantage of an adversary against the transparency is $Adv^{\mathcal{SS}}_{Trans} \leq Adv^{\mathcal{CH}}_{Uni} + Adv^{PRG}_{Pseudorand} + Adv^{PRF}_{Pseudorand}$.

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

– For the **Signer-Accountability**, there are two possibilities.
  1. The adversary uses a collision generated by the sanitizer, and thus successfully obtains a value x such that $\mathsf{TAG} = PRG(x)$. This is impossible under the one-wayness of the function PRG.
  2. The adversary uses a TAG she has constructed. To win the experiment, the adversary has to generate a collision on the chameleon hash function, which can happen with probability $Succ^{\mathcal{CH}}_{CollRes}$.

  As a consequence, the probability of success of the adversary $\mathcal{A}^{\mathcal{SS}}_{sig-Acc}$ is $Succ^{\mathcal{SS}}_{sig-Acc} \leq Succ^{PRG}_{OneWay} + Succ^{\mathcal{CH}}_{SCollRes}$.
– A successful adversary against the **Sanitizer-accountability** has to find a correct collision on a message m, using the Proof algorithm. As m necessary respects ADM, the signature s in σ is necessary a forge of the classical signature scheme. As a consequence, the probability of success against the Sanitizer-accountability is $Succ^{\mathcal{SS}}_{san-Acc} \leq Succ^{\mathcal{S}}_{EU-CMA}$. □

4 Model for Extended Sanitizable Signatures

4.1 Additional Features for Sanitizable Signatures

In this paper, we study in detail 4 additional features for sanitizable signature, from which 3 have been introduced in [7]. These restrictions are set by the signer and must be taken into account by the sanitizer.

– `LimitSet`: this feature permits the signer to force some admissible blocks of a signed message to be modified only into a predefined set of sub-messages. More precisely, during the Sign algorithm the signer may define for each admissible block i a set V_i of available sub-messages. Then, the sanitizer must use one element $\mathsf{m}'_i \in V_i$ during her sanitization of this block. For this purpose, we introduce the set $\mathcal{V} = \{V_i \subset \{0,1\}^{\ell_i} : i \in \mathsf{ADM}\}$. Each V_i defines the set for the modifications of the block m_i. If the signer does not want to restrict the sanitizer in her modifications of the block m_i, then $V_i = \{0,1\}^{\ell_i}$.
– `EnforceModif`: with this feature, the signer forces the sanitizer to modify similarly several admissible blocks. If one is modified by the sanitizer during the Sanitize procedure, she must use the same modification for the other admissible blocks designated by the signer. We introduce $\mathsf{cond}_m = \{\mathcal{S}_i \subset [1, 2^\ell] : \forall j \in \mathcal{S}_i, j \in \mathsf{ADM}$ and $\forall k \neq i, j \notin \mathcal{S}_k\}$. Each set in cond_m corresponds to a set of, at least, two admissible blocks which should be modified similarly. Note that an admissible block can only belong to one set $\mathcal{S}_i$.
– `LimitNbModif`: the sanitizer should modify less than a number k, fixed by the signer, out of the $|\mathsf{ADM}|$ admissible blocks. If the sanitizer modifies more than k blocks, one of her secret key becomes available. cond_k is a condition simply described by the integer $k \in [1, |\mathsf{ADM}|]$. In case this feature is not chosen by the signer, then $k = |\mathsf{ADM}|$.
– `LimitNbSanit`: this feature limits the number of versions one sanitizer can do from an original signed message. If the sanitizer does one extra sanitization, one of her secret key becomes available. We here define cond_l, which

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

corresponds to an integer l. If $l \neq \infty$, then the sanitizer can only sanitize the corresponding signed message l times. If $l = \infty$, then, there is no restriction on the number of sanitizations the sanitizer can do.

4.2 Modification of the Initial Model

We now modify the model of Brzuska *et al.* [2] to introduce the above new features. We first study the case of the classical procedures for sanitizable signature schemes Sign and Sanitize (as usual each of them output $\perp$ in case of error).

- Sign takes as input a message m of length ℓ divided into t blocks, the secret key sk_{sig}, the public key pk_{san} and ADM. It outputs a sanitizable signature σ on the message m and the variables $\mathcal{V}$, cond_m, cond_k and cond_l as defined above. This procedure may also output some secret data denoted s that would be needed by the sanitizer. We denote $\mathsf{s} = \perp$ if this is not relevant for the signer in the scheme. In the following, ADM is included into σ.
- Sanitize takes as input a message m, a signature σ, the keys pk_{sig} and sk_{san}, the modifications MOD and furthermore the variables $\mathcal{V}$, cond_m, cond_k and cond_l, as defined above, and the secret data s. It outputs a new signature σ', the message m' modified according to the different conditions and variables $\mathcal{V}$, cond_m, cond_k and cond_l defined by the signer.

Remark 1. Note that the different variables may not be used to verify the signature. In some cases, it may be necessary to keep these data secret. The verifier may e.g. not know how many times the sanitizer can sanitize a message. What is important is to detect a fraud, even if it may be simpler using the value l.

We now consider that the sanitization secret key sk_{san} is divided into two parts. The first one, usk_{san}, is considered as the user secret key and can be retrieved in case of fraud. It can be computed during a distinct procedure (e.g. some kind of UserKeyGen procedure) or included into the SanKeyGen phase. The second key, ssk_{san}, is used to sanitize messages, as the sanitization secret key in the initial model. We now introduce the new procedures.

- TestFraud is a public algorithm which on input the public keys pk_{sig}, pk_{san} and a set DB of pairs (message, signature), checks if a fraud has been done on the number of admissible blocks and/or on the number of versions of message. It outputs either $\perp$ is everything is ok, or usk_{san} and a proof of guilt π otherwise. We consider that π includes DB .
- VerifyFraud is a public algorithm which on input a proof π and a user key usk_{san} outputs either 1 if the proof π is valid, and 0 otherwise.

4.3 Relation Between Security Properties

We now focus on the security properties that need to be modified to take into account the above features. First of all, the accountability and the transparency properties are not modified by the above additional features. However the oracles

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

used in both experiment should be modified in order to consider the additional input. For example, in the transparency experiment, the SIGN/SANIT oracle generates a sanitized signature for all values of the challenge bit b and this oracle and the SIGN one should be honest together (i.e. they cannot make more than l sanitizations (for example) in total). Note that regarding the privacy property (implied by transparency [2]), we need to slightly modify the corresponding experiment to add in the LORSANIT oracle the choice of the $\mathcal{V}_i$'s: for each admissible part $k \in \mathsf{ADM}^+$, both initial and final messages should belong to the randomly chosen set $\mathcal{V}_k$. Thus, privacy is also induced by transparency in the extended model. We now concentrate on immutability.

Extended immutability. Let us focus on the two first additional extensions which modify the immutability property. In the new model, we add two new conditions to the classical experiment (i) one modifiable part m_i is not in the set of acceptable values for that part ($\mathsf{m}'_i \notin V_i$) or (ii) two admissible blocks from the same set element $\mathcal{S}_{i_0}$ of cond_m (that is that are forced to be modified accordingly) are different. Note that we have the following lemma.

Lemma 1. *An Extended Immutable signature scheme is Immutable.*

Proof. A successful adversary $\mathcal{A}$ against the immutability experiment [2] outputs $(\mathsf{pk}_{san}{}^*, \mathsf{m}^*, \sigma^*)$ such that $\textsc{Verify}(\mathsf{m}^*, \sigma^*, \mathsf{pk}_{sig}, \mathsf{pk}_{san}{}^*) = \mathsf{true}$ and for all $i = 1, 2, \cdots, q$ either $\mathsf{pk}_{san}{}^* \neq \mathsf{pk}_{san,i}$ or $\exists j_i \notin \mathsf{ADM}_i | \mathsf{m}^*[j_i] \neq \mathsf{m}_i[j_i]$, so she directly wins the Extended Immutability experiment above. □

Extended traceability. We now introduce a new security property which we call "extended traceability". This ensures that an adversary is not able to do more modifications than stated by SIGN, or to execute more sanitizations of the same signed message than the sanitizer is allowed to, without being accused of.

In the corresponding experiment, the adversary outputs several valid pairs (message, signature) under the same sanitizer public key such that TESTFRAUD, with as input this set of pairs, detects a fraud i.e. returns a pair $(\mathsf{usk}_{san}, \pi)$. The adversary wins the game if usk_{san} is not part of the corresponding sanitizer secret key or if VERIFYFRAUD outputs 0.

Extended Immutability: $Succ^{\mathcal{SS}}_{ext-imm} = Pr[1 \longleftarrow \textsc{Exp}^{\mathcal{SS}}_{ext-imm}]$ where $\textsc{Exp}^{\mathcal{SS}}_{ext-imm}$ is:
- $(\mathsf{pk}_{sig}, \mathsf{sk}_{sig}) \longleftarrow \textsc{SigKeyGen}(1^\lambda)$
- $(\mathsf{pk}^*_{san}, \mathsf{m}^*, \sigma^*, l^*, k^*, \mathcal{V}^*) \longleftarrow \mathcal{A}^{\textsc{Sign}(\cdot, \mathsf{sk}_{sig}, \cdot, \cdot), \textsc{Proof}(\mathsf{sk}_{sig}, \cdot, \cdot, \cdot, \cdot)}(\mathsf{pk}_{sig})$
- Let $(\mathsf{m}_i, \mathsf{ADM}_i, \mathsf{pk}_{san,i})$ and $(\sigma_i, \mathcal{V}_i, \mathsf{cond}_{m_i}, \mathsf{cond}_{k_i}, \mathsf{cond}_{l_i})$ for $i \in [1, q]$ be the queries and answers to and from oracle SIGN.
- return 1 if $\textsc{Verify}(\mathsf{m}^*, \sigma^*, \mathsf{pk}_{sig}, \mathsf{pk}^*_{san}) = \mathsf{true}$ and for all $i = 1, 2, \cdots, q$ we have
  - $\mathsf{pk}^*_{san} \neq \mathsf{pk}_{san,i}$ or
  - $\exists j_i \notin \mathsf{ADM}_i$ such that $\mathsf{m}^*[j_i] \neq \mathsf{m}_i[j_i]$ or
  - $\exists j$ such that $\mathsf{m}^*[j] \notin V^*_j$ or
  - $\exists i_0$ such that $\exists j, j' \in \mathcal{S}_{i_0}$ such that $\mathsf{m}^*[j] \neq \mathsf{m}^*[j']$.

Extended Traceability: $Succ^{\mathcal{SS}}_{ext-tra} = Pr[1 \longleftarrow \textsc{Exp}^{\mathcal{SS}}_{ext-tra}]$ where $\textsc{Exp}^{\mathcal{SS}}_{ext-tra}$ is:
- $(\mathsf{pk}_{sig}, \mathsf{sk}_{sig}) \longleftarrow \textsc{SigKeyGen}(1^\lambda)$
- $(\mathsf{pk}^*_{san}, \mathsf{DB}^* = \{(m^*_p, \sigma^*_p), p = 1, \cdots, n\}) \longleftarrow \mathcal{A}^{\textsc{Sign}(\cdot, \mathsf{sk}_{sig}, \cdot, \cdot), \textsc{Proof}(\mathsf{sk}_{sig}, \cdot, \cdot, \cdot, \cdot)}(\mathsf{pk}_{sig})$
- If it exists $p \in [1, n]$ such that $\textsc{Verify}(m^*_p, \sigma^*_p, \mathsf{pk}_{sig}, \mathsf{pk}^*_{san}) = \mathsf{false}$, then outputs 0
- $(\mathsf{usk}_{san}, \pi) \longleftarrow \textsc{TestFraud}(\mathsf{pk}_{sig}, \mathsf{pk}^*_{san}, \mathsf{DB}^*)$
- return 1 if usk_{san} does not correspond to pk^*_{san}, or
  - $\textsc{VerifyFraud}(\pi, \mathsf{usk}_{san}) = 0$.

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

5 Cryptanalysis of Extended Sanitization Scheme

In this section, we review the paper of Klonowski and Lauks [7] and show that their `EnforceModif` and `LimitNbModif` extensions are insecure.

5.1 The `EnforceModif` Extension

We first recall the proposal in [7]. We assume that the signer signs $\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_t$ with d blocks $\mathsf{m}_{i_1}, \cdots, \mathsf{m}_{i_d}$ such that the sanitizer can modify similarly.

- SIGN: the signer chooses at random $x_1, \cdots, x_t, r$ and computes, for all $i \in [1, t]$, $h_i = g^{x_i}$ with g a public value. Then, she computes $c = h_1^{\mathsf{m}_1} \cdots h_t^{\mathsf{m}_t} g^r$ and a classical signature s on c. Finally, the signature σ is $(c, r, s, h_1, \cdots, h_t)$ and the sanitizer is given a secret value $\mathsf{s} = x_{i_1} + \cdots + x_{i_d}$.
- SANITIZE: the sanitizer wants to modify the signed message $\mathsf{m}_1 \| \cdots \| \mathsf{m}_t$ into the new one $\mathsf{m}_1^* \| \cdots \| \mathsf{m}_t^*$, with $\mathsf{m} = \mathsf{m}_{i_1} = \cdots = \mathsf{m}_{i_d}$ and $\mathsf{m}^* = \mathsf{m}_{i_1}^* = \cdots = \mathsf{m}_{i_d}^*$. On input $(c, r, s, h_1, \cdots, h_t)$ and s, the sanitized signature is simply $(c, r^*, s, h_1, \cdots, h_t)$ where $r^* = r + (\mathsf{m} - m^*)\mathsf{s}$.
- VERIFY: from the signature $\sigma = (c, r, s, h_1, \cdots, h_t)$ and the message $\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_t$, one can verify that $c = h_1^{\mathsf{m}_1} \cdots h_t^{\mathsf{m}_t} g^r$ and that s on c is valid.

In fact, from two different versions, with m and m^* identically modified, one can retrieve s and thus sanitize any message. In fact, if these blocks are the only admissible ones, from the construction of the SANITIZE, we have $r^* = r + (\mathsf{m} - m^*)\mathsf{s}$. As r, m, r^* and m^* are included into the signatures or the messages and $\mathsf{m} \neq \mathsf{m}^*$, one can compute $\mathsf{s} = \frac{r - r^*}{\mathsf{m}^* - \mathsf{m}}$. Thus this scheme is not accountable.

5.2 The `LimitNbModif` Extension

The solution proposed in [7] is based on polynomial interpolation: there is exactly one polynomial of degree at most k going through $k + 1$ fixed points. Then, the principle is to define a secret polynomial F of degree k, such that the sanitizer key $\mathsf{usk}_{san} = F(0)$. Each time the sanitizer sanitizes a block, a point of the polynomial F leaks. Thus, when $k + 1$ blocks are modified, $k + 1$ points are available, the secret polynomial can be interpolated and the sanitizer's secret key is retrieved. In [7], the basic sanitizable signature scheme is the one of Ateniese *et al.* with a chameleon hash function not resistant to the key exposure attack. For each modified block during the SANITIZE procedure, a point on the polynomial is chosen as the used key. Thus, as soon as a block is modified, a collision is computed and the point leaks.

More precisely, their scheme works as follows. Let usk_{san} be the sanitizer secret key related to $\mathsf{upk}_{san} = g^{\mathsf{usk}_{san}}$. During the SIGN procedure, the sanitizer chooses at random k values $f_1, \cdots, f_k$ and constructs the polynomial $F(y) = \mathsf{usk}_{san} + f_1 y + \cdots + f_k y^k$. She next computes $\{g_i = g^{f_i}\}_{i \in [1,k]}$, where g is a public generator, and sends it to the signer. The signer computes a sanitizable public

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

key for each admissible block m_i as $z_i = g^{F(i)} = \mathsf{upk}_{san} \cdot g_1^i \cdot \ldots \cdot g_k^{i^k}$. She next chooses an identifier for the message ID_{m} and a random r_i for each admissible block. She uses the chameleon hash function and computes, for all i, $\tilde{\mathsf{m}}_i = \textsc{Proceed}(z_i, \bar{\mathsf{m}}_i = ID_{\mathsf{m}} \| \mathsf{m}_i \| i, r_i) = z_i^{\bar{\mathsf{m}}_i} \cdot g^{r_i} (= g^{F(i) \cdot \bar{\mathsf{m}}_i + r_i})$ if $i \in ADM$ and $\tilde{\mathsf{m}}_i = \mathsf{m}_i \| i$ otherwise. The signature is $\sigma = (ID_{\mathsf{m}}, t, \{r_i\}_{\forall i \in \mathsf{ADM}}, \{z_i\}_{\forall i \in \mathsf{ADM}}, s)$ with s a classical signature on $(ID_{\mathsf{m}} \| t \| \mathsf{upk}_{san} \| \tilde{\mathsf{m}}_1 \| \cdots \| \tilde{\mathsf{m}}_t)$.

During the Sanitize algorithm, for each block m_j the sanitizer wants to modify, she uses the secret key $F(j)$ and computes a collision on $\tilde{\mathsf{m}}_j$. As the function is weak against the key exposure attack, $F(j)$ necessary leaks. Thus if she modifies $k+1$ blocks, $k+1$ points of $F(y)$ leaks and anybody can find usk_{san}.

Again, this solution is not secure since, after one sanitization, the value $F(i)$ necessary leaks. As the knowledge of $F(i)$ is enough to construct any collision on $\tilde{\mathsf{m}}_i$, one can construct as many other sanitizations as she wants from only one sanitization: the scheme is not accountable.

6 Constructions in the Extended Model

6.1 The `LimitSet` Extension

This feature has been nicely solved in [7] by the use of accumulators. It consists in accumulating all possible modifications for a block into one accumulator. The sanitizer is given the accumulator, the accumulated values and the corresponding witnesses to prove that one value is truly accumulated. Then, the accumulator is signed by the signer as a non admissible part of the message. During the sanitization process, the sanitizer should have to give the accumulated value, which is the new message block, and the corresponding witness, so that the verifier can verify that the modified block is a valid message for the focused admissible block.

- SigKeyGen. This step executes the SigKeyGen procedure of the initial scheme, as described in Section 3. We thus obtain $(\mathsf{ssk}, \mathsf{spk})$ and a secret key κ in $\{0,1\}^\lambda$ for the PRF. Then it executes the initialisation algorithm of the chosen accumulator scheme Acc.
- SanKeyGen. This algorithm is identical to the initial SanKeyGen.
- Sign. Let $\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_t$ be the message to be signed. The signer first generates the variable ADM: she decides for each block $i \in [1, t]$ whether the block is admissible or not. There are then two cases:
  1. the i-th block is not admissible ($i \notin \mathsf{ADM}$). The signer sets $\tilde{\mathsf{m}}_i = \mathsf{m}_i \| i$.
  2. the i-th block is admissible ($i \in \mathsf{ADM}$). There are two new cases:
     (a) there are no restriction on the value for this block (we say that $i \in \mathsf{ADM}^-$). The signer next chooses at random a value denoted $\mathsf{r}_i \in \{0,1\}^\lambda$ and computes $\tilde{\mathsf{m}}_i = \mathcal{CH}.\textsc{Proceed}(\mathsf{pk}_{san}, \mathsf{m}_i \| i, \mathsf{r}_i)$.
     (b) the i-th message block should lie in a set of authorized values $\mathcal{V}_i$ defined by the signer (we say that $i \in \mathsf{ADM}^+$). In this case, the signer first initializes an empty accumulator Acc_i and, for each element $a_{k,i} \in \mathcal{V}_i$, she accumulates it in Acc_i and computes the corresponding

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

witness $w_{k,i}$. The set of all witnesses for the block i is denoted $\mathcal{W}_i = \{w_{k,i} : k \in [1, |\mathcal{V}_i|]\}$. Note that the value m_i necessary lies in $\mathcal{V}_i$ for obvious reasons. That is, it exists k_0 such that $\mathsf{m}_i = a_{k_0,i}$. At the end of this step, the signer defines $\tilde{\mathsf{m}}_i = \mathsf{Acc}_i$.

In the following, we denote $W_0 = \{w_{k_0,i} : i \in \mathsf{ADM}^+\}$ the set of all witnesses used by the signer for the message m, $\mathcal{A} = \{\mathsf{Acc}_i : i \in \mathsf{ADM}^+\}$, $\mathcal{W} = \{\mathcal{W}_i : i \in \mathsf{ADM}^+\}$ and $\mathcal{R} = \{\mathsf{r}_i : i \in \mathsf{ADM}^-\}$.

The signer generates $\mathsf{TAG} = \mathrm{PRG}(x)$ where $x = \mathrm{PRF}(\kappa, \mathsf{Nonce})$ with $\mathsf{Nonce} \in \{0,1\}^\lambda$. She computes $(\mathsf{h}_c, \mathsf{r}_c) = \mathcal{CH}.\textsc{Proceed}(\mathsf{pk}_{san}, \mathsf{TAG}||\mathsf{m}, \mathsf{r}_c)$ and signs the message $\tilde{\mathsf{m}} = \tilde{\mathsf{m}}_1||\cdots||\tilde{\mathsf{m}}_t||\mathsf{h}_c||\mathsf{pk}_{san}$ as $s = \mathcal{S}.\textsc{Sign}(\mathsf{sk}_{sig}, \tilde{\mathsf{m}})$. Finally, the signature is $\sigma = (s, \mathsf{TAG}, \mathsf{Nonce}, \mathcal{R} \cup \{\mathsf{r}_c\}, \mathsf{ADM}, W_0)$. The set of authorized values for each admissible block $\mathcal{V} = \{\mathcal{V}_i : i \in \mathsf{ADM}^+\}$ and the corresponding set of all witnesses $\mathcal{W}$ (if they are not publicly computable) are independently sent to the sanitizer.

- Sanitize. The sanitizer wanting to modify the message m to the message m' performs the following actions, for each block $j \in [1, t]$:
  1. the j-th block is not admissible, the sanitizer does not do anything.
  2. the j-th block is admissible. There are two new cases:
     (a) if $j \in \mathsf{ADM}^-$, she computes $\mathsf{r}'_j = \mathcal{CH}.\textsc{Forge}(\mathsf{sk}_{san}, \mathsf{m}_j\|j, \mathsf{m}'_j\|j, \mathsf{h}_j)$.
     (b) if $j \in \mathsf{ADM}^+$, the sanitizer checks that the new block message $\mathsf{m}'_j \in \mathcal{V}_j$ and finds the corresponding $w_{k_0,j}$ in the set of all witnesses.

  The sanitizer next sets $W'_0 = \{w_{k_0,j} : j \in \mathsf{ADM}^+\}$ the set of all used witnesses for the new message m' and by $\mathcal{R}' = \{\mathsf{r}'_i : i \in \mathsf{ADM}^-\}$.

  She next chooses at random Nonce' and TAG' and uses sk_{san} to find a collision on the chameleon hash for the obtained message. That is, she recomputes h_c and computes $\mathsf{r}'_c = \mathcal{CH}.\textsc{Forge}(\mathsf{sk}_{san}, \mathsf{TAG}||\mathsf{m}, \mathsf{TAG}'||\mathsf{m}', \mathsf{h}_c)$. The new sanitize signature is finally $\sigma' = (s, \mathsf{TAG}', \mathsf{Nonce}', \mathcal{R}' \cup \{\mathsf{r}'_c\}, \mathsf{ADM}, W'_0)$.
- Verify. The verifier executes the reconstruction procedure. For all i, she defines $\tilde{\mathsf{m}}_i$ as (i) $\textsc{Acc}(\mathsf{m}_i, w_i)$ if $\mathsf{m}_i \in \mathsf{ADM}^+$, (ii) h_i if $\mathsf{m}_i \in \mathsf{ADM}^-$ or (iii) $\mathsf{m}_i||i$ otherwise. Then the verifier computes $\mathsf{h}_c = \mathcal{CH}.\textsc{Proceed}(\mathsf{pk}_{san}, \mathsf{TAG}||\mathsf{m}, \mathsf{r}_c)$ and verifies whether $\mathcal{S}.\textsc{Verify}(\mathsf{pk}_{sig}, s, \tilde{\mathsf{m}})$ returns true or false.
- Proof and Judge are identical to our classical construction (cf. Section 3).

Theorem 2. *Our scheme is secure if the signature scheme is $EU-CMA$, PRG and PRF are pseudo-random, and $\mathcal{CH}$ is strong secure and* Acc *is secure.*

Proof. Our scheme is ext-immutable, transparent, accountable and ext-traceable.

- In the **Ext-Immutability** experiment, $\mathcal{A}$ outputs $(\mathsf{pk}^*_{san}, \mathsf{m}^*, \sigma^*, l^*, k^*, \mathcal{V}^*)$. Either $\mathsf{pk}^*_{san} \neq \mathsf{pk}_{san,i}$ or $\exists j_i \notin \mathsf{ADM}_i$ such that $\mathsf{m}^*[j_i] \neq \mathsf{m}_i[j_i]$. That cases lies on the chosen signature EU-CMA property, similarly as in the proof of Immutability of our classical scheme (cf. Section 3). Or $\exists j$ such that $\mathsf{m}^*[j] \notin V^*_j$. In that case, either $\mathsf{m}^*[j]$ has been added to the accumulator of the i-th block and we can construct an adversary $\mathcal{A}^{\mathcal{S}}_{EU-CMA}$ against the EU-CMA property in outputting the forgery on the message $\tilde{\mathsf{m}}^*$. Or the adversary has find a value which has not been accumulated and we are able to construct an adversary $\mathcal{A}^{\mathsf{Acc}}_{CR}$ against the collision resistance of Acc. The success probability is finally $Succ^{\mathsf{set-SS}}_{Ext-Imm} \leq Succ^{\mathcal{S}}_{EU-CMA} + Succ^{\mathrm{Acc}}_{CR}$.

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

– For the **Transparency** property, we remark that an original sanitizable signature is $(s, \mathsf{TAG}, \mathsf{Nonce}, \mathcal{R} \cup \{\mathsf{r}_c\}, \mathsf{ADM}, W_0)$, while a sanitized one is $(s, \mathsf{TAG}', \mathsf{Nonce}', \mathcal{R}' \cup \{\mathsf{r}'_c\}, \mathsf{ADM}, W'_0)$. As the used witnesses (in W_0 and W'_0) are constructed in the same way for an original or a sanitized signature, the transparency property relies on the classical parts of the signature and the proof is identical to the proof in the classical case. The advantage of an adversary is $Adv^{\mathsf{set-SS}}_{Trans} \leq Adv^{\mathcal{CH}}_{Uni} + Adv^{PRG}_{Pseudorand} + Adv^{PRF}_{Pseudorand}$.
– Both **Accountability** properties rely on the construction of the last block of our construction h_c: it depends on the construction of TAG and on the incapability of the signer to obtain collisions on the chameleon hash. As accumulators are not implied on this part of the protocol, the proof is identical as in the classical case. Thus, $Succ^{set-\mathcal{SS}}_{sig-Acc} \leq Succ^{PRG}_{OneWay} + Succ^{\mathcal{CH}}_{SCollRes}$ and $Succ^{\mathsf{set-SS}}_{san-Acc} \leq Succ^{\mathcal{S}}_{EU-CMA}$. □

6.2 The `LimitNbModif` Extension

As in [7], our solution uses polynomial interpolation and a chameleon hash function CH weak against the key exposure attack. However, contrary to [7], we use a second *secure* chameleon hash function $\mathcal{CH}$. Thus, the sanitization phase on the message m_i requires the user to know both $\{F(i)\}_{i\in\mathsf{MOD}}$ (keys for CH) and the sanitizer secret key sk_{san} (for $\mathcal{CH}$). Thus, the leakage of $F(i)$ does not compromise the unforgeability any more.

More precisely, let usk_{san} be the sanitizer secret key, related to the public key $\mathsf{upk}_{san} = g^{\mathsf{usk}_{san}}$.

– Sign. The signer executes the signature procedure as for our initial scheme and obtains ADM, the value $\tilde{\mathsf{m}}$, TAG, Nonce and $\mathcal{R} = \{\mathsf{r}_i\}_{i\in\mathsf{ADM}}$. Then she randomly chooses $\mathcal{R}^* = \{\mathsf{r}^*{}_i\}_{i\in\mathsf{ADM}}$. Meanwhile, the sanitizer randomly chooses k values $f_1, \cdots, f_k$ and constructs $F(y) = \mathsf{usk}_{san} + f_1 y + \cdots + f_k y^k$. She next computes the set $\{g_i = g^{f_i}\}_{i\in[1,k]}$, where g is a public generator, and sends it to the signer. After that, the signer computes the set $\{z_i\}_{i\in\mathsf{ADM}}$ such that $z_i = \mathsf{upk}_{san} \cdot g_1^i \cdot \ldots \cdot g_k^{i^k} (= g^{F(i)})$ and uses each z_i as the public key of CH to hide the corresponding r_i: $\forall i \in \mathsf{ADM}, t_i = CH.\textsc{Proceed}(z_i, \mathsf{r}_i, \mathsf{r}^*_i) = z_i^{\mathsf{r}_i} \cdot g^{\mathsf{r}^*_i} (= g^{F(i)\cdot \mathsf{r}_i + \mathsf{r}^*_i})$. Finally, she classically signs the concatenation of $\tilde{m}$ and the $\{t_i\}_{i\in\mathsf{ADM}}$: $\bar{s} = \mathcal{S}.\textsc{Sign}(\mathsf{sk}_{sig}, \tilde{m} \| t_{i_1} \| \cdots \| t_{i_{|\mathsf{ADM}|}})$ and obtains the signature $\bar{\sigma} = \{\bar{s}, TAG, \mathsf{Nonce}, \mathcal{R}, \mathcal{R}^*, \{z_i\}_{i\in\mathsf{ADM}}, \mathsf{ADM}\}$.
– Sanitize. The sanitizer computes, for each block m_j she wants to modify, a collision on $\tilde{\mathsf{m}}_j$ thanks to sk_{san} and a collision on t_j thanks to $F(j)$.

Remark 2. Note that the Sign process is suppose to be non-interactive. In the above description, we can imagine than the sanitizer regularely publishes some g_i's that can be used by the signer when necessary. In some other cases, such as for content protection [5], the sanitizer is considered as on line during this process, and thus can compute on line these g_i's.

With this method, the sanitizer can easily modify k blocks. As in [7], if she modifies $k+1$ blocks, $k+1$ points of $F(y)$ are available and usk_{san} leaks. But

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

in our case, the collision resistance of $\mathcal{CH}$ already fixes the message. thus it does not impact the security of the scheme any more.

Theorem 3. *Our scheme is secure if the signature scheme is EU-CMA, PRG and PRF are pseudo-random, $\mathcal{CH}$ (resp. CH) is strong (resp. weak) secure.*

Proof. Our scheme is extended immutable, transparent and accountable for the same reasons than our main scheme. For the ext-traceability there are two cases. Either, the adversary outputs $(\mathsf{pk}^*_{san}, \mathsf{DB}^* = \{(m^*_p, \sigma^*_p), p = 1, \cdots, n\})$ such that pk^*_{san} does not correspond to usk_{san} output by TestFraud. In our scheme, this is checked by TestFraud. So the underlying success probability is 0. Or, $\mathcal{A}$ outputs $(\mathsf{pk}^*_{san}, \mathsf{DB}^* = \{(m^*_p, \sigma^*_p), p = 1, \cdots, n\})$ such that TestFraud detects a fraud $(\mathsf{usk}_{san}, \pi)$ and that VerifyFraud outputs 0. This may happens either if usk_{san} is not the secret key corresponding to the public key of the sanitizer, but this has already been studied, or if the polynomial $F(y)$ obtained through interpolation is such that $g^{F(0)} \neq \mathsf{upk}_{san}$. As only one polynomial of degree n goes through $n+1$ given point, and considering that the public key is signed, an adversary able to modify this value can be used to construct an adversary against the EU-CMA of the chosen signature. So the success probability is $Succ^{\mathcal{SS}}_{Ext-trans} \leq Succ^{\mathcal{S}}_{EU-CMA}$. □

6.3 The `LimitNbSanit` Extension

In a nutshell, our system is based on a method which has been first proposed for the e-cash purpose. It consists in using the soundness property of zero-knowledge proofs of knowledge of a secret. Honest-verifier proofs are three-move protocols: a commitment t based on random values, a question c and an answer s related to the above random values, the question and the secret. The soundness of these constructions ensures that given a single t, if someone is able to provide s and s' related to c and c' *s.t.* $c \neq c'$, then it is possible to retrieve the secret.

More precisely, our methodology works as follows. First, we use our main sanitizable signature scheme described in Section 3. Let usk_{san} be the sanitizer secret key, related to the public key $\mathsf{upk}_{san} = g^{\mathsf{usk}_{san}}$.

- Sign. The sanitizer chooses at random l values $a_1, \cdots, a_l$. She next computes, for all $i \in [1, l]$, the value $t_i = g^{a_i}$, with g a public generator. Each value t_i corresponds to a version number authorized by the signer. The sanitizer next sends $\{t_i\}_{1 \leq i \leq l}$ to the signer. After that, the signer chooses two random values α and ρ and constructs its own version number: $t_0 = \mathsf{upk}^{\rho}_{san} g^{\alpha}$. Then the signer accumulates all the t_i's (including t_0) into one single accumulator Acc and executes the Sign procedure of our main sanitizable signature scheme, using ρ as random for the final execution of the chameleon hash function $\mathcal{CH}$ on the whole message, and adding Acc in the final classical signature: she obtains σ. The signature is $\bar{\sigma} = \{\sigma, t_0, \alpha, \mathsf{Acc}, w_0\}$ with w_0 the witness for t_0. Finally, the witnesses of the accumulated values are given to the sanitizer as the secret s.

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.

– SANITIZE. The sanitizer executes the SANITIZE procedure of main scheme. Then she reveals a new t_i and its corresponding witness, denoted w_i, her public key upk_{san} and the value $\alpha_i = a_i - \rho_i \mathsf{usk}_{san}$ with ρ_i the pseudorandom value output during the generation of the collision on the whole message.

With this method, the sanitizer can easily use the l different accumulated values. However, if the sanitizer executes $l+1$ times this procedure, she has to use twice the same accumulated value with her secret key usk_{san} but with two different random values ρ_i and ρ_j. It is thus possible to retrieve usk_{san}.

Theorem 4. *Our scheme is secure if the signature scheme is EU-CMA, PRG and PRF are pseudo-random and $\mathcal{CH}$ is strong secure and* ACC *is secure.*

Proof. Our scheme is ext-immutable, transparent and accountable for the same reasons than our main scheme. The **Ext-Traceability** implies that $\mathcal{A}$ outputs DB^* under pk^*_{san} such that TESTFRAUD finds a fraud. In fact, $\mathcal{A}$ should either embed more values in Acc, which breaks the security of ACC, or is able to output a signature on a wrong accumulator and thus breaks the EU-CMA of the signature scheme. In conclusion, $Succ^{\mathcal{SS}}_{Ext-trans} \leq Succ^{\text{ACC}}_{CR} + Succ^{\mathcal{S}}_{EU-CMA}$. □

References

1. G. Ateniese, D. H. Chou, B. De Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS 2005*, volume 3679 of *LNCS*, pages 159–177. Springer.
2. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of sanitizable signatures revisited. In *PKC 2009*, volume 5443 of *LNCS*, pages 317–336. Springer.
3. J. Camenisch, M. Kohlweiss, and C. Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credential. In *PKC 2009*, volume 5443 of *LNCS*, pages 481–500. Springer.
4. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer.
5. S. Canard, F. Laguillaumie, and M. Milhau. Trapdoor sanitizable signatures and their application to content protection. In *ACNS 2008*, volume 5037 of *LNCS*, pages 258–276. Springer.
6. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17:281–308, 1988.
7. M. Klonowski and A. Lauks. Extended sanitizable signatures. In *ICISC 2006*, volume 4296 of *LNCS*, pages 343–355. Springer.
8. H. Krawczyk and T. Rabin. Chameleon signatures. In *NDSS 2000*. The Internet Society.
9. L. Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA 2005*, volume 3376 of *LNCS*, pages 275–292. Springer.
10. T. H. Yuen, W. Susilo, J. K. Liu, and Y. Mu. Sanitizable signatures revisited. In *CANS 2008*, volume 5339 of *LNCS*, pages 80–97. Springer.

Appeared in J. Pieprzyk (Ed.): CT-RSA 2010, volume 5985 of LNCS, pp. 179–194.