Trapdoor Sanitizable Signatures and Their Application to Content Protection*

Sébastien Canard[1], Fabien Laguillaumie[2], and Michel Milhau[1]

[1] Orange Labs R&D, 42 rue des Coutures, BP6243, F-14066 Caen Cedex, France
[2] GREYC Université de Caen, Campus 2, Boulevard du Maréchal Juin, BP 5186, 14032 Caen Cedex, France

Abstract. Sanitizable signatures allow a designated entity to modify some specific parts of a signed message and to produce a new signature of the resulting message without any interaction with the original signer. In this paper, we extend these sanitizable signatures to formally introduce *trapdoor sanitizable signatures*. In this concept, the power of sanitization is given to possibly several entities, for a given message/signature by using a trapdoor computed by the signer at any time. We also give a generic construction of such trapdoor sanitizable signatures. Eventually, we apply our new cryptographic tool to group content protection, permitting members of the group to distribute a protected content among themselves.

1 Introduction

Digital Rights Management (DRM) systems provide efficient mechanisms to protect digital contents against unauthorized usages. Despite its increasing usage, in particular in video and music on-line services, users have difficulties to agree with device limitations induced by these systems. Consumers want flexibility with *legal* transfer or exchange of their acquired digital contents.

Let us consider the following scenario: a modern family made up of two parents, teenagers and children connected to a physical or wireless LAN. This family wants to buy some music songs offered by a well-known music service portal. The father identifies itself and purchases the desired protected media content on the portal of the license server. His local DRM agent installs in his device a license associated to this musical content. A license is a data structure which contains information about the authorized rights on the song and the cryptographic key necessary for decrypting the content. For security reasons this license has been strongly tied to the device. A license is protected by cryptographic functions: a part of it is encrypted with the public key of the recipient's device and consequently, it is the only device able to decrypt information helpful for a content use. In this situation, other family members could play this song only on the father's device.

* This work has been financially supported by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT.

S.M. Bellovin et al. (Eds.): ACNS 2008, LNCS 5037, pp. 258–276, 2008.

This is a main drawback in spite of a few advances made in this field. Some private DRM applications allow the transfer of limited rights up to non connected devices like Personal Digital Assistant (i.e. Microsoft WMDRM). The normalization group OMA-DRM[1] itself defined the notion of domain to resolve exchange of rights/licenses inside a set of mobile devices belonging to a same domain. In fact each domain's device receives the same cryptographic key allowing the use of the domain license. The management of these domains and the generation of the associated keys are realized by the license server. This means that this server gets a specific knowledge about domain's members.

Similarly, iTunes from Apple enables buyers to transfer content to identified machines that share the same license decryption key. The decryption key is sent to a new computer from Apple's servers that can, by this way, limit the number of authorized computers (today limited to five).

To by-pass these restrictions, we propose a system which answers the following needs: (1) dynamicity of license transfers; (2) local management of the group members; (3) compliance with OMA-DRM architecture and protocols. The first and second requirements are incompatible with OMA-DRM domains; in fact we need to modify the former license received by the family's father. The DRM agent of the father will be able to create a new license targeted at another family member. This modification of license requires both operations: (1) decrypt the Content Encryption Key CEK thanks to the father's private key, and encrypt it again with the public key of the new recipient, (2) sign the new license such that the DRM agent of the recipient recognizes this signature as a true license server signature.

Technically, to get rid of this drawback, we need to adapt and develop a new type of signature scheme where it is possible for a designated user to modify some part of a message signed by a particular signer. The signature on the new message is still seen as a message signed by the initial signer. In DRM systems, the signer of a license is the license server and applying the new signature scheme, the result is that the OMA-DRM agent of the final user will accept the signature of the derived license as a real license server signature. Moreover, it should be possible to give this particular power later (*i.e.* not at the creation of the license) to permit users to change their mind when they want to.

1.1 Related Work

A usual way to expose declassified documents and keep a protection of intelligence sources is to blot out the sensitive parts. Well-known examples were the memo to US President that had been declassified for an inquiry into the 11 September 2001 terrorist attack and a US Department of Defense memo about who helped Iraq to militarize civilian Hugues helicopters. These examples of sanitized documents became famous because Naccache and Whelan demasked the blotted out words [13].

[1] Open Mobile Alliance is a standard consortium which develops open standards for the mobile phone industry.

In this example, keeping secret the sanitized part is fundamental. When applied to electronic documents, sanitizing can of course easily resist Naccache and Whelan attack, but an important issue is the *authentication* and *integrity* of such sanitized documents. If the original document is signed with a traditional signature scheme, modifying this document in any way will make the original signature be invalid. If the authentication must be preserved, a traditional signature cannot be used in this case. Of course, the signer could sign the sanitized document, but when applied to declassified documents for instance, the signer's secret key might have expired, or the signer may not be available at all. *Sanitizable signatures* have been introduced to address this problem.

Two different flavors of sanitizable signatures can be found in the literature. The first kind illustrates the previous scenario, namely the signature of the original message can be modified, without the help of the signer, to be also valid for the sanitized document [16,7,15]. In other words, a designated user can erase some part of a signed message and produce a new signature in such a way that the resulting signature is seen as a correct signature on the new message from the initial signer.

The other kind of sanitizable signatures was introduced in [1] by Ateniese, Chou, de Medeiros and Tsudik, also in terms of sanitizable signatures. Contrary to the above case, such signatures allow a semi-trusted censor to *modify* (not only to erase) some specific portions of a signed message and to produce a new valid signature of the resulting message without any interaction with the original signer. According to Ateniese *et al.*, a sanitizable signature scheme must ensure (1) *immutability*, which means that the censor must not be able to modify any part of the message, not specified by the signer, (2) *privacy*, which means that all sanitized information is unrecoverable, (3) *accountability*, which means that in case of dispute, the signer can prove to the court that a given message was sanitized, and (4) *transparency* which means that no one, except the signer and the censor, can guess whether a message has been sanitized. As we will see in section 3.1, Ateniese *et al.*'s scheme can easily be obtained *via* a sanitizable signature scheme of the first kind.

In [9], Klonowski and Lauks propose some improvement of the Ateniese et al. scheme [1] by proposing (in particular) generic techniques permitting first to limit the set of possible modifications of a single mutable block using either accumulator schemes or bloom filters and second to limit the number of modifications of mutable blocks. Note that their methods cannot be applied to our scheme since they do not use a trapdoor.

1.2 Our Contribution and Organization of the Paper

The sanitizable signature of Ateniese *et al.* may be useful for our application of content protection. But in fact, the proposed properties of their sanitizable signature scheme are not sufficient. More precisely, we need the possibility for any authorized user to have a special trapdoor to modify some designated parts of a signed message. Therefore, contrary to the modelization of Ateniese *et al.*, the power of the sanitizer is not given during the signature process.

In this paper, we thus define in Section 2 the security model of a new type of sanitizable signatures, called *trapdoor* sanitizable signatures, where the power of sanitization is given to possibly several entities, for a given message/signature by using a trapdoor issued by the signer at any time.

We also give the first generic construction in Section 3 of such sanitizable signature scheme based on an identity-based chameleon hashing function and on a classical signature scheme. We then give a possible instance of our generic construction and apply in Section 4 our new cryptographic tool to content protection so as to fit our requirement for a DRM license, as described previously.

2 Trapdoor Sanitizable Signature Scheme

In a trapdoor sanitizable signature scheme, the signer allows a specific user to modify a portion of a signed message by producing a piece of information that will help this user in sanitizing the document. This trapdoor information is given upon the will of the signer, who can choose *to whom* and *when* he will deliver it. This last property makes a crucial difference with conventional sanitizable signatures, and is of importance to design our DRM scheme.

In this section, we first briefly and informally describe sanitizable signatures as they appear in the literature, and then, we propose a formal definition and a precise security model for our new primitive of *trapdoor* sanitizable signatures.

2.1 Review of Existing Definitions for Sanitizable Signatures

We recall here several definitions for sanitizable signature schemes to enlight the different notions and the difference with our new concept.

- According to [16,7,11,12,15], a sanitizable signature scheme is a scheme which allows a specific user (not necessarily chosen by the signer) to sanitize certain portions of a message (which means that these portions become unavailable) and to generate a valid signature for this new document, without any interaction with the signer. The scheme consists of four procedures: the key generation, the signing process, the sanitizing phase, and the verification. This scheme must resist an existential forgery under a chosen message attack, and must be indistinguishable under a chosen message attack (which means that an attacker is not able to distinguish between two signatures, which one is one a sanitized message).
- According to [1,9], a sanitizable signature scheme permits a specific user to modify a message (*a priori* chosen by the signer) and to produce a valid signature for the new message. The main difference with the previous definition is therefore the possibility to replace some parts of the documents with some others. The different algorithms which constitute the scheme are the same as in the previous definition. The scheme must be existentially unforgeable under a chosen message attack, and the authors also propose a notion of indistinguishability and one of *identical distribution* between the distribution coming from the signing algorithm and the one coming from the sanitizing algorithm.

2.2 Definition of a Trapdoor Sanitizable Signature Scheme

We present in this section a formal definition for *trapdoor sanitizable signatures* and its security model. The main difference with Ateniese *et al.*'s definition is that the original signer produces a trapdoor information, depending on a specific message divided into several blocks of independent sizes, that he will transmit to the sanitizer, who will be able to *modify* this message.

Definition 1 (Trapdoor sanitizable signature scheme). *A* trapdoor sanitizable signature scheme *TSS consists in the following algorithms.*

- *Setup is a probabilistic algorithm which takes a security parameter k as input and outputs the* public parameters $\mathcal{P}$. $\mathcal{P} \longleftarrow \mathsf{Setup}(k)$.
- *KeyGen is a probabilistic algorithm which takes the public parameters $\mathcal{P}$ as input and outputs a pair of secret and public keys (sk, pk).* $(sk, pk) \longleftarrow \mathsf{KeyGen}(\mathcal{P})$.
- *Sign is a probabilistic algorithm which takes public parameters $\mathcal{P}$, a message $m = m_1 \| \cdots \| m_L$ and a secret key sk as inputs, and outputs a signature σ on the message m and the set $I \subset [\![1, L]\!]$ of the indices that are sanitizable on this signature.* $(\sigma, I) \longleftarrow \mathsf{Sign}(\mathcal{P}, m, sk)$.
- *Trapdoor is a deterministic algorithm which takes public parameters $\mathcal{P}$, a message m and a valid signature σ and a secret key sk as inputs and outputs a trapdoor* $\mathbf{t}$. $\mathbf{t} \longleftarrow \mathsf{Trapdoor}(\mathcal{P}, m, \sigma, sk)$.
- *Sanitize is an algorithm which takes public parameters $\mathcal{P}$, a message m, a valid signature σ on m under the public key pk, a message $\tilde{m}$, the set I of the indices that are sanitizable and a trapdoor* $\mathbf{t}$ *and outputs a signature $\tilde{\sigma}$ on the message $\tilde{m}$.* $\tilde{\sigma} \longleftarrow \mathsf{Sanitize}(\mathcal{P}, m, \sigma, pk, \tilde{m}, I, \mathbf{t})$.
- *Verif is a deterministic algorithm which takes public parameters $\mathcal{P}$, a message m, a putative signature σ, a public key pk and the set I of the indices that are sanitizable as inputs and outputs* 1 *is the signature σ on m is valid and* 0 *otherwise.* $0/1 \longleftarrow \mathsf{Verif}(\mathcal{P}, m, \sigma, pk, I)$.

Remark 1. Note that the Sanitize algorithm can be either deterministic or probabilistic. In fact, in all existing constructions (including ours), as the construction is based on the use of chameleon hash functions, this algorithm is deterministic. But we can easily imagine that it is possible to design (trapdoor) sanitizable signature schemes with a probabilistic Sanitize algorithm.

The security criteria which must be fulfilled are discussed in the following paragraph.

2.3 Security Model

Correctness. A trapdoor sanitizable signature scheme must satisfy two *correctness* properties:

$$\forall k \in \mathbb{N}, \forall \mathcal{P} \leftarrow \mathsf{TSS.Setup}(k), \forall (pk, sk) \leftarrow \mathsf{TSS.KeyGen}(\mathcal{P}),$$
$$\forall (\sigma, I) \leftarrow \mathsf{TSS.Sign}(\mathcal{P}, m, sk), \mathsf{Verif}(\mathcal{P}, m, \sigma, pk, I) \rightarrow 1$$

and

$$\forall k \in \mathbb{N}, \forall \mathcal{P} \leftarrow \mathsf{TSS.Setup}(k), \forall (pk, sk) \leftarrow \mathsf{TSS.KeyGen}(\mathcal{P}),$$
$$\forall (\sigma, I) \leftarrow \mathsf{TSS.Sign}(\mathcal{P}, m, sk), \forall \mathbf{t} \leftarrow \mathsf{Trapdoor}(\mathcal{P}, m, \sigma, sk),$$
$$\forall \tilde{\sigma} \leftarrow \mathsf{Sanitize}(\mathcal{P}, m, \sigma, pk, \tilde{m}, I, \mathbf{t}), \mathsf{Verif}(\mathcal{P}, \tilde{m}, \tilde{\sigma}, pk, I) \rightarrow 1$$

Unforgeability. A trapdoor sanitizable signature scheme must also satisfy an unforgeability property. The conventional notion of security for signatures was introduced by Goldwasser, Micali and Rivest in [6]. A signature scheme must be *existentially unforgeable* under a *chosen message attack*. We present here the formal definition of existential unforgeability under a chosen message attack (EU-CMA) for trapdoor sanitizable signatures.

First, we need to define the following oracles.

- $\mathcal{O}^{\mathsf{Sign}}$: this oracle is initialized with a public key pk and the corresponding secret signing key sk. It takes as input a message m and outputs a valid signature related to this message and the public key pk.
- $\mathcal{O}^{\mathsf{Trapdoor}}$: this oracle is initialized with a public key pk and the corresponding secret signing key sk. It takes as input a message m and a signature σ. If $\mathsf{Verif}(\mathcal{P}, m, \sigma, pk, I) = 0$, then the oracle outputs `error`. Otherwise, it outputs a trapdoor $\mathbf{t}$ related to the message m, the signature σ and the public key pk as if it is output by the $\mathsf{Trapdoor}$ algorithm.
- $\mathcal{O}^{\mathsf{Sanitize}}$: this oracle is initialized with a public key pk and the corresponding secret signing key sk. It takes as input two messages m and $\tilde{m}$ and a signature σ. If $\mathsf{Verif}(\mathcal{P}, m, \sigma, pk, I) = 0$, then the oracle outputs `error`. Otherwise, it computes a trapdoor $\mathbf{t}$ related to the message m, the signature σ and the public key pk (using sk) and outputs the signature $\tilde{\sigma}$ on $\tilde{m}$ as if it is output by the $\mathsf{Sanitize}$ algorithm on input m, σ and $\tilde{m}$. Note that the trapdoor $\mathbf{t}$ can be either deleted or not at the end of the request.

We say that a trapdoor sanitizable signature scheme is *existentially unforgeable under a chosen message attack*, if no PPT adversary $\mathcal{F}$ has a non negligible success in the following game:

1. The challenger $\mathcal{C}$ runs the Setup algorithm to produce the public parameters and then runs the KeyGen algorithm. It obtains the pair of keys $(pk^\star, sk^\star)$ to be attacked and gives the public key $pk^\star$ to $\mathcal{F}$.
2. The forger $\mathcal{F}$ adaptively interacts with the signing oracle $\mathcal{O}^{\mathsf{Sign}}$ and the trapdoor oracle $\mathcal{O}^{\mathsf{Trapdoor}}$.

3. Eventually, $\mathcal{F}$ comes up with a message $m^\star$ and a signature $\sigma^\star$. $\mathcal{F}$ is said to *succeed* if the pair $(m^\star, \sigma^\star)$ verifies the four following properties.
   (a) $\mathsf{Verif}(\mathcal{P}, m^\star, \sigma^\star, pk^\star, I) \rightarrow 1$.
   (b) $(m^\star, \sigma^\star)$ does not come from the $\mathcal{O}^{\mathsf{Sign}}$ oracle.
   (c) $(m^\star, \sigma^\star)$ does not come from the $\mathcal{O}^{\mathsf{Sanitize}}$ oracle.
   (d) $(m^\star, \sigma^\star)$ is not linked to a tuple $(\mathbf{t}, m, \sigma)$ from the $\mathcal{O}^{\mathsf{Trapdoor}}$ oracle. More precisely, for all message m being in input of the $\mathcal{O}^{\mathsf{Trapdoor}}$ oracle, we should have that $\exists i \notin I_m, m_i^\star \neq m_i$ where I_m corresponds to the set I output with the signature σ and related to m.

Remark 2. Note that by describing condition (3d) like that, we reject the possibility for the adversary to forge a signature on a message related to one for which a trapdoor has been asked. This can be seen as restrictive (we don't know if the adversary has used the trapdoor or not to produce the forge) but this is not really a problem in practice. Note also that if the $\mathsf{Sanitize}$ algorithm is deterministic (see above), this is easily verifiable since the corresponding signature can be also computed by the challenger.

The *success* of $\mathcal{F}$ is defined as the probability (over all internal random coins) of its success in this previous game. $\mathcal{F}$ is said to $(q_S, q_T, q_{Sz}, \tau, \varepsilon)$-breaks the existential unforgeability in the chosen message attack of the trapdoor sanitizable signature scheme, if its success is ε, its running time is τ, and its number of queries to the signing oracle (resp. trapdoor oracle and sanitize oracle) is q_H (resp. q_T and q_{Sz}).

Indistinguishability. We require that values produced by the $\mathsf{Sanitize}$ algorithm are distributed identically to those produced by the Sign algorithm. In particular, the following distributions $\mathcal{D}_{\mathsf{Sanit}}$ and $\mathcal{D}_{\mathsf{Sign}}$ are indistinguishable for all $\mathcal{P}$, pk, sk:

$$\mathcal{D}_{\mathsf{Sanit}} = \{\tilde{\sigma} : \tilde{\sigma} = \mathsf{Sanitize}(\mathcal{P}, m, \sigma, pk, \tilde{m}, I, \mathbf{t}), (m, \tilde{m}) \in \mathcal{M}, \\ (\sigma, I) = \mathsf{Sign}(\mathcal{P}, m, sk), \mathbf{t} = \mathsf{Trapdoor}(\mathcal{P}, m, \sigma, sk)\}$$

and

$$\mathcal{D}_{\mathsf{Sign}} = \{\sigma : (\sigma, I) = \mathsf{Sign}(\mathcal{P}, m, sk), m \in \mathcal{M}\}.$$

3 Generic Construction

Before describing a generic construction of a trapdoor sanitizable signature scheme, we will first give an intuition of this construction by revisiting a bit Ateniese *et al.*'s scheme from [1] in the light of a sanitizable scheme of Miyazaki *et al.* [11,12]. As we already explained, the corresponding two definitions of sanitizable signatures are different. Nevertheless, the next section shows how Ateniese *et al.*'s scheme can be obtained from Miyazaki *et al.*'s one.

3.1 Ateniese *et al.*'s Sanitizable Signatures Revisited

Let's first recall the scheme SUMI-4 from Miyazaki *et al.*'s [11,12]. The rough idea is to replace the parts of the message that are censored by hash values of these parts. Let $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$ be a signature scheme and h be a hash function. SUMI-4 works as follows.

Signer

1. Let $m = m_1 \| \dots \| m_L$ be the message to sign, the signer first picks random values $r_i \in \{0,1\}^k$ for $i = 1 \dots L$.
2. The signer generates a set $H = \{h_i\}_{i=1\dots L}$ where $h_i = h(m_i, r_i)$
3. The signature is obtain as $\sigma = \mathsf{Sign}(H)$.
4. The signer outputs $(\{m_i, r_i)\}_{i=1\dots L}, \sigma)$

Sanitizer

1. Determine $I \subset [\![1, L]\!]$ the indices of the message to be censored.
2. The sanitizer converts the document m to $\tilde{m} = \tilde{m}_1 \| \dots \| \tilde{m}_L$ where

$$\tilde{m}_i = \begin{cases} (m_i, r_i) \text{ if } i \notin I \\ h_i \text{ if } i \in I \end{cases}$$

3. The sanitizer outputs $(\tilde{m}, I)$

Verifier

1. Let $(\tilde{m}, I)$ the sanitized message and σ the original signature. The verifier generates the set of hash values $\tilde{H} = \{\tilde{h}_i\}_{i=1\dots L}$ such that

$$\tilde{h}_i = \begin{cases} h(\tilde{m}_i) \text{ if } i \notin I \\ \tilde{m}_i \text{ if } i \in I \end{cases}$$

2. The verifier then checks $\mathsf{Verif}(\tilde{H}, \sigma)$ to verify the validity of the signature.

Now, let's look at this scheme from Ateniese *et al.*'s point of view. In their definition of sanitizable signatures, a specific user can not only erase a part of the message, but he can replace it by something else. Suppose that h is replaced by a *chameleon hash function*, as introduced by Krawczyk and Rabin in [10]. This means that a pair of secret and public key parametrizes this hash function, with the property that the owner of the secret key is able to find a collision to a hash value computed thanks to his public key. In this setting, the signer *chooses* the sanitizer (by selecting his "chameleon" public key) and replaces the function h by the chameleon hash function $h^{\mathcal{C}}$. The sanitizer is then able to replace the sanitized parts by values of his choice, thanks to his secret key. Indeed, given a message m'_i (for $i \in I$), he can compute an element r'_i such that

$$h^{\mathcal{C}}(m'_i, r'_i) = h^{\mathcal{C}}(m_i, r_i) = h_i.$$

The remaining works as follows: the sanitizer converts the document m to $\tilde{m} = \tilde{m}_1 \| \ldots \| \tilde{m}_L$ where

$$\tilde{m}_i = \begin{cases} (m_i, r_i) \text{ if } i \notin I \\ (m'_i, r'_i) \text{ if } i \in I \end{cases}$$

and the verifier generates the set of hash values $\{\tilde{h}_i\}_{i=1\ldots L}$ such that $\tilde{h}_i = h^{\mathcal{C}}(\tilde{m})$ and eventually checks $\mathsf{Verif}(H', \sigma)$ to verify the validity of the signature.

This is another look at Ateniese *et al.*'s scheme. Our idea to design trapdoor sanitizable signature is to replace the chameleon hash function by an *identity-based* chameleon hash function. The secret key extraction from an identity will actually help us to generate the trapdoor allowing the sanitization of a specific message. The following section rigorously describes our construction relying on this idea.

3.2 A Generic Construction of *Trapdoor* Sanitizable Signatures

We propose in this section a generic construction which uses as building blocks an identity-based chameleon hash function [10] to achieve our "trapdoor" requirement. Roughly speaking a chameleon hash function is a trapdoor hash function, such that the owner of the trapdoor is able to find collisions for every given input. When they are used instead of traditional hash functions in a signature scheme based on the well-known *hash-and-sign* paradigm, the resulting scheme is a *chameleon signature scheme* which is highly related to Chaum and van Antwerpen's *undeniable signatures* [5] and Jakobsson, Impagliazzo and Sako's *designated verifier signatures* [8].

Identity-Based Chameleon Hashing. Identity-based chameleon hashing were introduced by Ateniese and de Medeiros [2]. We assume that it is possible to identify all systems to a bit-string easily derivable from a system's public knowledge. We call such a string an identity string and we note it Id. There are two actors in such schemes: an authority $\mathcal{A}$ and a user $\mathcal{U}$. For mally, an *identity-based chameleon hash scheme* CH is defined by the following family of efficiently computable algorithms.

- Setup: this probabilistic algorithm is executed by $\mathcal{A}$ to generate a pair of key sk_{CH} and pk_{CH}, taking on input a security parameter k.
$$(sk_{CH}, pk_{CH}) \longleftarrow \mathsf{Setup}(k)$$

- $\mathsf{Extract}$: this probabilistic algorithm, executed by $\mathcal{A}$ taking on inputs the identity Id of a user $\mathcal{U}$ and the secret key sk_{CH}, outputs the derived trapdoor information B.
$$B \longleftarrow \mathsf{Extract}(Id, sk_{CH})$$

- $\mathsf{Proceed}$: this probabilistic algorithm that can be executed by anybody and which takes on inputs the public key pk_{CH}, the identity Id of a user, a message m and a random value r and outputs the hash value h.
$$h \longleftarrow \mathsf{Proceed}(pk_{CH}, Id, m, r)$$

– Forge: this algorithm is executed by the user with identity Id to compute a hash value on a new message. It takes as inputs the public key pk_{CH}, the identity Id, the corresponding extracted value B, a new message m' and the hash value h on a message m with random value r and it outputs a new hash value (h, r'). $r' \longleftarrow \mathsf{Forge}(pk_{CH}, Id, B, m', r, h, m)$

This construction must be *correct*, which means that if $B = \mathsf{Extract}(Id, sk_{CH})$ and if we assume that $h = \mathsf{Proceed}(pk_{CH}, Id, m, r)$ where m is a message and r a random value, then $h = \mathsf{Proceed}(pk_{CH}, Id, m', \mathsf{Forge}(pk_{CH}, Id, B, m', r, h, m))$.

We need to introduce the following new security property for the identity-based hash function, to prove the security of our scheme. This notion is not defined in Krawczyk and Rabin's paper [10] nor in the one of Ateniese et de Meideros [2], but it is a natural generalization of the notion of collision for traditional hash functions. An Id-based chameleon hash function must be *collision-resistant*, in the sense of the following game:

1. A challenger $\mathcal{C}$ runs the Setup algorithm to produce the pair of keys $(pk^\star, sk^\star)$ and gives the public key $pk^\star$ to $\mathcal{F}$.
2. The collision finder $\mathcal{F}$ adaptively interacts with an Extract oracle $\mathcal{O}^{\mathsf{Extract}}$ to obtain a trapdoor information B corresponding to some identity string.
3. Eventually, $\mathcal{F}$ comes up with a tuple (m, m', r, r', Id). $\mathcal{F}$ is said to *succeed* if $\mathsf{Proceed}(pk^\star, Id, m, r) = \mathsf{Proceed}(pk^\star, Id, m', r')$ and Id has not been proposed to the Extract oracle.

Moreover, it is required that the distributions of r and r' must be the same (random and uniform) and that a message m induces the same probability distribution on $\mathsf{Proceed}(pk^\star, Id, m, r)$ for a random r (*cf.* Krawczyk and Rabin's *uniformity* from [10]).

The Construction. Our construction also relies on a classical signature scheme Σ = (Setup,KeyGen,Sign,Verif) which follows the "hash-and-sign" paradigm. That is, from a key pair (sk_S, pk_S), output by the Σ.KeyGen algorithm, and a message m, the signature σ is computed as $\sigma = \Sigma.\mathsf{Sign}(sk_S, m)$. Eventually, from a signature σ, a message m and a public key pk_S, the verifier checks that $\Sigma.\mathsf{Verif}(pk_S, m, \sigma) = 1$. This signature scheme must be existentially unforgeable under a chosen message attack as defined in [6].

– Setup: the Setup consists in executing Σ.Setup to output some public parameters.
– KeyGen: the KeyGen phase consists in executing the $\mathsf{CH.Setup}(k)$ for the chameleon hash function and the $\Sigma.\mathsf{KeyGen}(k)$ for the signature scheme. Consequently, the global secret key of the trapdoor sanitizable scheme is $sk = (sk_{CH}, sk_S)$ and the corresponding public key is $pk = (pk_{CH}, pk_S)$.

Procedure $\mathsf{KeyGen}(k)$:

$(sk_{CH}, pk_{CH}) \longleftarrow \mathsf{CH.Setup}(k)$
$(sk_S, pk_S) \longleftarrow \Sigma.\mathsf{KeyGen}(k)$
$sk = (sk_{CH}, sk_S)$
$pk = (pk_{CH}, pk_S)$
Output (sk, pk)

– Sign: the Sign step first consists in choosing the set I of the indices that are sanitizable. For each $i \in I$, we execute $\mathsf{CH.Proceed}(pk_{CH}, m, m_i, r_i)$ which outputs h_i. For all $i \in [\![1, L]\!]$, we set $\hat{m}_i = m_i$ if $i \notin I$ and $\hat{m}_i = h_i$ otherwise. We denote by $\hat{m} = \hat{m}_1 \| \cdots \| \hat{m}_L$. σ is the concatenation of the output s of the $\Sigma.\mathsf{Sign}$ algorithm on input sk_S and $\hat{m}$ and of all elements of the set R. We also add a verification value h_c so as to prevent some types of attacks. The output of the algorithm is finally σ and the set I.

Procedure $\mathsf{Sign}(\mathcal{P}, m, sk)$:

$m_1 \| \cdots \| m_L \longleftarrow m$
Set $I \subset [\![1, L]\!]$
$\forall i \in I, r_i \in_R \mathcal{R}$
$\forall i \in I, h_i \longleftarrow \mathsf{CH.Proceed}(pk_{CH}, m, m_i, r_i)$
$\forall i \in [\![1, L]\!] \setminus I, \hat{m}_i \longleftarrow m_i$
$\forall i \in I, \hat{m}_i \longleftarrow h_i$
$r_c \in_R \mathcal{R}$
$h_c \longleftarrow \mathsf{CH.Proceed}(pk_{CH}, m, m, r_c)$
$\hat{m} = \hat{m}_1 \| \cdots \| \hat{m}_L \| h_c$
$R \longleftarrow \{r_i : i \in I\}$
$\sigma \longleftarrow \Sigma.\mathsf{Sign}(sk_S, \hat{m})$
$\forall i \in I, \sigma \longleftarrow \sigma \| r_i$
$\sigma \longleftarrow \sigma \| r_c$
Output (σ, I)

– Trapdoor: the Trapdoor function consists in executing the CH.Extract of the Id-based chameleon hash function with m as the identity and sk_{CH}. It outputs the derived trapdoor information $\mathbf{t}$.

Procedure $\mathsf{Trapdoor}(\mathcal{P}, m, \sigma, sk)$:

if $\mathsf{Verif}(\mathcal{P}, m, \sigma, pk, I) = 1$ then
  $\mathbf{t} \longleftarrow \mathsf{CH.Extract}(m, sk_{CH})$
  Output $\mathbf{t}$
otherwise output `error`

– Sanitize: let $\tilde{m} = \tilde{m}_1 \| \cdots \| \tilde{m}_L$. Remember that σ is the concatenation of s and all elements of $R = \{r_i : i \in I\}$. The CH.Forge algorithm is executed for all $i \in I$ on input pk_{CH}, m, $\mathbf{t}$, the new message $\tilde{m}_i$, r_i, h_i and m_i that

outputs each time a new $\tilde{r}_i$ and we do the same for the verification value h_c. The new signature $\tilde{\sigma}$ is thus the concatenation of s (unchanged) and all elements of $\tilde{R} = \{\tilde{r}_i : i \in I\}$ plus r_c.

Procedure $\mathsf{Sanitize}(\mathcal{P}, m, \sigma, pk, \tilde{m}, I, \mathbf{t})$:

$m_1 \| \cdots \| m_L \longleftarrow m$
$\tilde{m}_1 \| \cdots \| \tilde{m}_L \longleftarrow \tilde{m}$
Retrieve s, R and r_c from σ
$\forall i \in I, h_i \longleftarrow \mathsf{CH.Proceed}(pk_{CH}, m, m_i, r_i)$
$\forall i \in I, \tilde{r}_i \longleftarrow \mathsf{CH.Forge}(pk_{CH}, m, \mathbf{t}, \tilde{m}_i, r_i, h_i, m_i)$
$\tilde{R} \longleftarrow \{\tilde{r}_i : i \in I\}$
$h_c \longleftarrow \mathsf{CH.Proceed}(pk_{CH}, m, m, r_c)$
$\tilde{r}_c \longleftarrow \mathsf{CH.Forge}(pk_{CH}, m, \mathbf{t}, \tilde{m}, r_c, h_c, m)$
$\sigma \longleftarrow s$
$\forall i \in I, \sigma \longleftarrow \sigma \| \tilde{r}_i$
$\sigma \longleftarrow \sigma \| \tilde{r}_c$
Output σ

– Verif: the verification procedure consists in computing h_i for each m_i with $i \in I$ by using $\mathsf{CH.Proceed}$ on input pk_{CH}, $m = m_1 \| \cdots \| m_L$, m_i and r_i. For all $i \in [\![1, L]\!]$, we set $\hat{m}_i = m_i$ if $i \notin I$ and $\hat{m}_i = h_i$ otherwise and we denote by $\hat{m} = \hat{m}_1 \| \cdots \| \hat{m}_L \| h_c$. The output of the Verif algorithm is the output of $\Sigma.\mathsf{Verif}(pk_S, s, \hat{m}) = 1$.

Procedure $\mathsf{Verif}(\mathcal{P}, m, \sigma, pk, I)$:

Retrieve s, $R = \{r_i : i \in I\}$ and r_c from σ
$m_1 \| \cdots \| m_L \longleftarrow m$
$\forall i \in I, h_i \longleftarrow \mathsf{CH.Proceed}(pk_{CH}, m, m_i, r_i)$
$\forall i \in [\![1, L]\!] \setminus I, \hat{m}_i \longleftarrow m_i$
$\forall i \in I, \hat{m}_i \longleftarrow h_i$
$h_c \longleftarrow \mathsf{CH.Proceed}(pk_{CH}, m, m, r_c)$
$\hat{m} = \hat{m}_1 \| \cdots \| \hat{m}_L \| h^c$
Output $\Sigma.\mathsf{Verif}(pk_S, s, \hat{m})$

3.3 Security

First of all, the correctness of our scheme is obvious. We will then concentrate our security analysis on unforgeability and indistinguishability.

Theorem 1 (Unforgeability)
Let $\mathcal{F}$ be a $(q_S, q_T, q_{Sz}, \tau, \varepsilon)$-forger against our trapdoor sanitizable signature scheme. Then their exists a $(\varepsilon', \tau', q_S)$-existential forger $\mathcal{F}'$ against the underlying signature scheme and a $(\varepsilon'', \tau'', q_T + q_{Sz})$-collision finder $\mathcal{C}$ against the identity-based hash function and a such that:

$$\varepsilon \leq \frac{1}{2}(\varepsilon' + \varepsilon'') \text{ and } \tau \geq \max\{\tau' + (q_T + q_{Sz})t_{\mathsf{collision}}, \tau'' + q_S t_{\mathsf{sign}}\}$$

where $t_{\mathsf{collision}}$ and t_{sign} are the necessary time to compute a collision and to sign a message respectively.

Proof. To simplify the proof (and in particular to get rid of the set I of indices in the proof), we will suppose that a message consists of a single block[2]. The condition (3d) of acceptation of the forge by the adversary needs then to be modified, taking into account that our Sanitize algorithm is deterministic. More precisely, considering the forge $(m^\star, \sigma^\star)$ output by the adversary, we can easily test, for all message (m, σ) for which the adversary has asked the $\mathcal{O}^{\mathsf{Trapdoor}}$ oracle, whether or not the Sanitize algorithm, on input $m, \sigma, m^\star, \mathbf{t}$ output $\sigma^\star$. If this is the case, the output of the adversary is rejected.

Let's suppose that their exists a $(q_S, q_T, q_{Sz}, \tau, \varepsilon)$-forger against our trapdoor sanitizable signature scheme. Given a public key pk as input, after at most q_S queries to the signing oracle, q_T queries to the trapdoor oracle, and q_{Sz} queries to the sanitize oracle, the forger outputs a pair $m^\star, \sigma^\star$, with $\sigma^\star$ corresponding to a random coin $r^\star$. For the forger to win the game, two possibilities arise:

- case 1: $m^\star$ is a non-sanitized message and $\sigma^\star$ has not been obtained from the signing oracle
- case 2: $m^\star$ is a sanitized message which has not been formed with a trapdoor information obtained from the trapdoor oracle nor the sanitize oracle. The pair $(\sigma^\star, r^\star)$ must not come from the signing oracle. Nevertheless, we can suppose that $\sigma^\star$ comes from the signing oracle.

We will show that the first case allows the construction of a forger against the signature scheme Σ, the second allows the construction of a collision-finder on the identity-based hash function CH. We will exhibit a reduction $\mathcal{R}$ which will flip a coin $b \in \{0, 1\}$ to bet which case will happen.

Case 1. Let's first consider the case 1 where the reduction designs an existential forger $\mathcal{F}'$. $\mathcal{R}$ will use $\mathcal{F}$ as a sub-routine to build this forger, which has as input a public key $pk^\star$ and some global parameters $\mathcal{P}$, and has to produce an existential forgery related to this public key. First, $\mathcal{R}$ generates a pair of key (pk_{CH}, sk_{CH}) for the identity-based chameleon hash function thanks to CH.Setup. Then he sets $pk = (pk^\star, pk_{CH})$ and gives this public key to $\mathcal{F}$. $\mathcal{R}$ now has to simulate $\mathcal{F}$'s signing, trapdoor and sanitize oracles. To simulate the trapdoor and the sanitize oracles $\mathcal{O}^{\mathsf{Trapdoor}}$ and $\mathcal{O}^{\mathsf{Sanitize}}$, the reduction $\mathcal{R}$ uses its knowledge of the trapdoor secret key sk_{CH} to create the trapdoor (and, in case of $\mathcal{O}^{\mathsf{Sanitize}}$, computes the corresponding signature). To simulate the signing oracle $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{R}$ must answer to $\mathcal{F}$'s query related to a message m. It forwards this query to $\mathcal{F}'$'s signing oracle which is parametrized by the secret key sk, and therefore

[2] However, this do not prevent an adversary to create new signatures without the knowledge of the trapdoor. The proof can be generalizable but will not be detailled in this paper due to space constraints.

$\mathcal{R}$ simulates perfectly $\mathcal{F}'s$ environment. At the end of the game, $\mathcal{F}$ outputs a pair $(m^\star, \sigma^\star)$ (with corresponding random bits $r^\star$). $\mathcal{R}$ then sets $(m^\star, \sigma^\star)$ as $\mathcal{F}'$'s outputs. This is clearly a successful forgery.

Case 2. Now, we consider that the reduction designs a collision-finder $\mathcal{C}$ against the identity-based chameleon hash function CH from case 2. The reduction $\mathcal{R}$ has to deal with $\mathcal{C}$'s challenge : namely a public key $pk^\star_{CH}$ for the chameleon hash function. $\mathcal{R}$ then executes $\Sigma.\mathsf{Setup}$ and $\Sigma.\mathsf{KeyGen}$ to obtain the system's global parameters and a pair of keys (sk_Σ, pk_Σ) for the signature scheme Σ. As before, $\mathcal{R}$ constructs a public key for the trapdoor sanitizable signature scheme $pk = (pk^\star_{CH}, pk_\Sigma)$, which he forwards to the forger $\mathcal{F}$. To simulate the signing oracle, the reduction $\mathcal{R}$ uses the secret key sk_Σ. When the forger $\mathcal{F}$ queries a trapdoor for a pair (m, σ), $\mathcal{R}$ first checks the validity of the signature with the public key pk_Σ and then asks $\mathcal{C}$'s extractor oracle $\mathcal{O}^{\mathsf{Trapdoor}}$ with m as the (identity) request. $\mathcal{O}^{\mathsf{Trapdoor}}$'s answer is a trapdoor $\mathbf{t}$ which corresponds to a correct trapdoor for sanitization. When the forger $\mathcal{F}$ queries a sanitization for a message m and its signature σ, $\mathcal{R}$ first checks the validity of the signature with the public key pk_Σ, secondly asks $\mathcal{C}$'s extractor oracle $\mathcal{O}^{\mathsf{Trapdoor}}$ with m as the (identity) request to obtain the corresponding trapdoor $\mathbf{t}$ and finally computes the new signature using the trapdoor. The reduction therefore perfectly simulates $\mathcal{F}$'s environment.

At the end of this game, $\mathcal{F}$ comes up with a pair $(m^\star, \sigma^\star)$ corresponding to a random string $r^\star$. As we suppose that $m^\star$ is a sanitized message, their exists a message $\tilde{m}^\star$, corresponding to a signature $(\sigma^\star, \tilde{r}^\star)$, which has been asked to the signing oracle, such that

$$\mathsf{CH.Proceed}(pk_{CH}, m^\star, m^\star, r^\star) = \mathsf{CH.Proceed}(pk_{CH}, m^\star, \tilde{m}^\star, \tilde{r}^\star).$$

Eventually, $(m^\star, \tilde{m}^\star, r^\star, \tilde{r}^\star, m^\star)$ is a collision for the chameleon hash function. The final success probability and running time are straightforward, and this concludes the proof. □

Theorem 2 (Indistinguishability). *The following distributions are perfectly indistinguishable for all $\mathcal{P}$, pk, sk:*

$$\mathcal{D}_{\mathsf{Sanit}} = \{\tilde{\sigma} : \tilde{\sigma} = \mathsf{Sanit}(\mathcal{P}, m, \sigma, pk, \tilde{m}, I, \mathbf{t}), (m, \tilde{m}) \in \mathcal{M}, \\ (\sigma, I) = \mathsf{Sign}(\mathcal{P}, m, sk), \mathbf{t} = \mathsf{Trapdoor}(\mathcal{P}, m, sk)\}$$

and

$$\mathcal{D}_{\mathsf{Sign}} = \{\sigma : (\sigma, I) = \mathsf{Sign}(\mathcal{P}, m, sk), m \in \mathcal{M}\}.$$

Proof. Let's first consider $\mathcal{D}_{\mathsf{Sign}}$. As described in the previous section, the output of Sign consists in a signature σ and a set I. The signature σ is composed of the outputs of $\Sigma.\mathsf{Sign}$ and a random (uniformly chosen) value $r_i \in \mathcal{R}$ where Sign is a classical signature scheme. Let us denotes $\mathsf{Im}(\Sigma.\mathsf{Sign})$ the distribution of all outputs of the Sign algorithm of the chosen signature scheme. Consequently, $\mathcal{D}_{\mathsf{Sign}} = \{(\sigma, r_i) : \sigma \in \mathsf{Im}(\Sigma.\mathsf{Sign}), r_i \in \mathcal{R}\}$.

Let us now consider $\mathcal{D}_{\mathsf{Sanit}}$. As described in the previous section in the description of the $\mathsf{Sanitize}$ algorithm, the signature $\tilde{\sigma}$ is composed of the value σ such that $(\sigma, I) = \mathsf{Sign}(\mathcal{P}, m, sk)$ and a value $\tilde{r}_i$ that is output by a call to the $\mathsf{CH.Forge}$ algorithm. As explained above, it implies that $\sigma \in \mathsf{Im}(\Sigma.\mathsf{Sign})$ and that σ is on the same distribution $\mathsf{Im}(\Sigma.\mathsf{Sign})$ as the previous one. Moreover, as shown in the previous section in useful tool, a secure identity-based chameleon hashing function has the property that $\mathsf{Im}(\mathsf{CH.Forge})$ is the set of all $r_i \in \mathcal{R}$ informally distributed and thus the set $\mathcal{R}$. Consequently,

$$\mathcal{D}_{\mathsf{Sanit}} = \{(\sigma, r_i) : \sigma \in \mathsf{Im}(\Sigma.\mathsf{Sign}), \tilde{r}_i \in \mathcal{R}\}.$$

These two distributions are obviously indistinguishable in the sense of the information theory which concludes the proof. □

3.4 An Example of Instantiation

In [2], the authors propose the following construction of an Id-based chameleon hash function.

- $\mathsf{CH.Setup}$: $n = pq$ is an RSA modulus where p and q are of size the security parameter k, v is a random prime element and w corresponds to the inverse of v modulo $\varphi(n)$.Then, $sk_{CH} = (p, q, w)$ and $pk_{CH} = (n, v)$.
- $\mathsf{CH.Extract}$: from the secret key sk_{CH} and an identifier Id, it is possible to construct the extracted key by first computing $J = \mathtt{EMSA} - \mathtt{PSS}(Id)$ and $\mathbf{t} = J^w \pmod n$.
- $\mathsf{CH.Proceed}$: from the public key pk_{CH}, the identifier Id and a message m, choosing at random r, it is possible to compute $J = \mathtt{EMSA} - \mathtt{PSS}(Id)$ and the value h is then $h = J^{\mathcal{H}(m)} r^v \pmod n$ where $\mathcal{H}$ is a classical hash function.
- $\mathsf{CH.Forge}$: this algorithm, taking on input pk_{CH}, Id, $\mathbf{t}$, a new message m', r, h and m outputs the random value r' associated with m' and h. This is done by $r' = r\mathbf{t}^{\mathcal{H}(m) - \mathcal{H}(m')} \pmod n$.

We can thus use this instantiation of an Id-based chameleon hash function in our proposal. The proof of Theorem 1 of [2] can be easily modified to prove that the scheme reaches the collision resistance property as defined previously in this paper. Other choices for secure identity-based chameleon hash function can be found in [17].

A lot of classical signature schemes can be used for our generic construction and one possible choice is RSA with EMSA-PSS padding [14].

4 Application to Group Content Protection

4.1 Classical Approach and Limitations

As defined in OMA DRM, a DRM agent $\mathcal{A}$ needs an encryption key pair (sk_a, pk_a) to interact with a License Server $\mathcal{L}$ which will provide him a license. The License Server needs a signature key pair (sk_l, pk_l). A protected content is always encrypted with a unique secret key denoted by CEK. A license is related to a single protected content denoted by Id_c. It consists in the following fields:

- the license identifier Id_l which is a unique serial number of the license,
- the content identifier Id_c,
- the rights which describe what is possible to do (read, copy, etc.) with this content using this license,
- the content key CEK encrypted for the receiver using pk_a,
- the signature σ from the License Server of all the previous fields.

Consequently, the license can be viewed as the concatenation of a message $m = Id_l \| Id_s \| \mathsf{rights} \| [CEK]_{pk_a}$ and a signature σ.

There are several steps to describe the way a DRM agent is able to read a protected content in the classical approach. These are the following ones.

- SystemCreation: the License Server generates his signature keys.

$$(sk_l, pk_l) \longleftarrow \mathsf{SystemCreation}(\mathcal{P}).$$

  In the OMA DRM standard, it is recommended to use the RSA signature scheme with an EMSA-PSS padding.
- AgentCreation: a DRM agent generates his encryption keys.

$$(sk_a, pk_a) \longleftarrow \mathsf{AgentCreation}(\mathcal{P}).$$

  The OMA DRM standard recommends to use the RSA encryption scheme.
- ContentEncapsulation: during this phase, the License Server encrypts the content C using a randomly generated secret key CEK. The encrypted content is then published.

$$PC \longleftarrow \mathsf{ContentEncapsulation}(CEK, C).$$

- LicenseGen: this is an interactive protocol between a DRM agent $\mathcal{A}$ and the License Server $\mathcal{L}$ where $\mathcal{A}$ first sends to $\mathcal{L}$ the chosen content Id_c and its encryption public key pk_a. $\mathcal{L}$ then creates the license $L = (m, \sigma)$ as described above and sends it back to $\mathcal{A}$.

$$L \longleftarrow \mathsf{LicenseGen}(Id_l, Id_c, \mathsf{rights}, CEK, sk_l, pk_a).$$

- ContentRead: the DRM agent retrieves the protected content and the license, verifies the signature of the License Server on the license, verifies the rights, decrypts CEK using its private decryption key and finally decrypts the content using CEK.

$$C \longleftarrow \mathsf{ContentRead}(PC, L, sk_a, pk_l).$$

On one hand, the problem is that the license is completely related to the DRM agent and this latter cannot send a received license to other DRM agents. In fact, this security property is very useful in many cases since it prevents fraud, but in the context of a group of DRM agents which wants to buy some contents and then to share them, this may be a too much important restriction.

On the other hand we do not want to modify *the way a DRM agent proceeds* when reading a protected content. Consequently, it is necessary to design a way to transfer a license to another DRM agent without modifying the structure of a license nor the signature from the License Server. A license bought from the License Server and a license coming from another DRM agent should have the same structure, including the same signature. By this way, constructors need to implement only one type of DRM agent. But, for this purpose, we need first to introduce some new procedures to the ones described above.

4.2 Some New Procedures

We use the same model as the one of classical OMA DRM one we describe above except that the License Server needs a new key pair (sk, pk) related to the computation of a trapdoor that will permit a designated DRM agent to modify a designated license so that other chosen DRM agents will be able to use the license. We thus add the new following procedures.

- TrapdoorGen: this is an interactive protocol between a DRM agent $\mathcal{A}$ and the License Server $\mathcal{L}$ where the latter gives to $\mathcal{A}$ the possibility to modify some part of a specific license.

$$\mathbf{t} \longleftarrow \mathsf{TrapdoorGen}(L, sk).$$

- TransferLicense: a DRM agent sends to another one a license. This latter can be used classically by the receiver using the ContentRead procedure.

$$\tilde{L} \longleftarrow \mathsf{TransferLicense}(L, \mathbf{t}, \widetilde{pk}_a, pk).$$

Note that we want to give the buyer a maximum of flexibility. This implies the possibility for her to choose, whenever she wants, a classical license with no possibility of transfer and later the possibility to transfer a previously obtained license. This is done by using our trapdoor sanitizable signature scheme as described in the next section. Note also that this is for this particular reason that the Ateniese et al. [1] proposal of sanitizable signature is not suitable here.

4.3 General Description

In our proposal, we maintain previous known procedures as they are. We do not modify the SystemCreation, AgentCreation, ContentEncapsulation, LicenseGen and ContentRead procedures. The only modification concerns the signature scheme since we do not use a classical RSA signature scheme but a trapdoor sanitizable signature scheme based on RSA with EMSA-PSS padding.

Consequently, the SGen call in the SystemCreation procedure is replaced in the OMA DRM standard by the execution of the KeyGen of a trapdoor sanitizable signature scheme that outputs (sk, pk).

Moreover, in the LicenseGen protocol, the Σ.Sign algorithm is replaced by an execution of the TSS.Sign of the trapdoor sanitizable signature scheme. Consequently, a license has the same fields as a classical one except that the signature

from the License Server is not a calssical signature. Notice however that the underlying signature scheme is standard since and we can use the RSA signature scheme as a building block (as a hash and sign type signature scheme). The verification of the signature done by a DRM Agent when trying to read a content, initially done by the Σ.Verif algorithm of a classical signature scheme is also replaced by a call to the TSS.Verif algorithm of the trapdoor sanitizable signature scheme. Let us now describe the two new procedures.

- TrapdoorGen: this is done by executing the TSS.Trapdoor algorithm of the trapdoor sanitizable signature scheme with the public parameters, the license L and the secret key sk of the trapdoor sanitizable signature scheme as inputs. The output is the trapdoor $\mathbf{t}$.
- TransferLicense: a DRM Agent having a valid license and a trapdoor $\mathbf{t}$ can execute the TransferLicense procedure by executing the TSS.Sanitize algorithm of the trapdoor sanitizable signature scheme with the initial license L (containing the message M and the signature σ), the trapdoor sanitizable signature public key pk, the new message $\tilde{m}$ that corresponds to the different fields (except the signature) of the new license, the corresponding set I of indices in the message that are sanitizable (see Section 4.4 below) and the trapdoor $\mathbf{t}$ as inputs. The output is a signature $\tilde{\sigma}$ which is concatenated to the message $\tilde{m}$ to create the new license $\tilde{L}$.

4.4 On the Modification of a License

As described above, it is possible for a DRM Agent to modify some fields of a valid license using the trapdoor sanitizable signature scheme properties. We consider that a license is a (reduced) message m and the signature σ of the License Server. The message m is divided into several blocks: $m_1 = Id_l$, $m_2 = Id_c$, $m_3 =$ rights and $m_4 = [CEK]_{pk_a}$. In the following, we study which parts of the license are sanitizable.

The messages m_1 and m_2 should of course not be modified. On the contrary, the message m_4 will be modified by the DRM Agent since the receiver should be able to decrypt the CEK to read the content. Thus the value 4 necessary belongs to the set I that is output by the TSS.Sign algorithm.

The case of the rights is a bit more complicated and there are several possibilities. Either it is not possible for the DRM Agent to modify the rights, or it is possible but only in a set of predefined values (at least no more rights than the ones the DRM Agent already has). In the first case, our trapdoor sanitizable signature scheme can be used as it is. In the second case, we need to modify it and it is an open problem to adapt the techniques of [9] to our scheme.

5 Conclusion

We formally introduce a new variant of sanitizable signatures and apply our new tool to manage licenses for digital contents protection within a group. We hope

that our new variant can also be useful for medical applications or secure routing (see [1]). Among some open problems, we suggest to add a traitor tracing layer or to have a better control on the message that can be sanitized.

Acknowledgments. We are grateful to anonymous referees for their valuable comments.

References

1. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable Signatures. In: di Vimercati, S.d.C., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 159–177. Springer, Heidelberg (2005)
2. Ateniese, G., de Medeiros, B.: Identity-based chameleon hash and application. In: Juels, A. (ed.) FC 2004. LNCS, vol. 3110, pp. 164–180. Springer, Heidelberg (2004)
3. Ateniese, G., de Medeiros, B.: On the Key Exposure Problem in Chameleon Hashes. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 165–179. Springer, Heidelberg (2005)
4. Bellare, M., Rogaway, P.: The exact security of digital signatures: How to sign with RSA and Rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)
5. Chaum, D., van Antwerpen, H.: Undeniable Signatures. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 212–216. Springer, Heidelberg (1990)
6. Goldwasser, S., Micali, S., Rivest, R.L.: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. SIAM J. Comput. 17(2), 281–308 (1988)
7. Izu, T., Kanaya, N., Takenaka, M., Yoshioka, T.: PIATS: A Partially Sanitizable Signature Scheme. In: Qing, S., Mao, W., López, J., Wang, G. (eds.) ICICS 2005. LNCS, vol. 3783, pp. 72–83. Springer, Heidelberg (2005)
8. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated Verifier Proofs and Their Applications. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 143–154. Springer, Heidelberg (1996)
9. Klonowski, M., Lauks, A.: Extended Sanitizable Signatures. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296, pp. 343–355. Springer, Heidelberg (2006)
10. Krawczyk, H., Rabin, T.: Chameleon Signatures. In: Proc. NDSS 2000, pp. 143–154. The Internet Society (2000)
11. Miyazaki, K., Iwamura, M., Matsumoto, T., Sakai, R., Yoshiura, H., Tezuka, S., Imai, H.: Digitally Signed Document Sanitizing Scheme with Disclosure Condition Control. IEICE Trans. on Fundamentals E88-A(1), 239–246 (2005)
12. Miyazaki, K., Susaki, S., Iwamura, M., Matsumoto, T., Sasaki, R., Yoshiura, H.: Digital Documents Sanitizing Problem. IEICE technical report, ISEC 2003-20 (2003)
13. Naccache, D., Whelan, C.: 9/11: Who Alerted the CIA (And Other Secret Secrets). In: Eurocrypt 2004 rump session (2004)
14. RSA Labs. RSA Cryptography Standard: EMSAPSS – PKCS#1 v2.1 (2002)
15. Steinfeld, R., Bull, L., Zheng, Y.: Content Extraction Signatures. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 286–304. Springer, Heidelberg (2002)
16. Suzuki, M., Isshiki, T., Tanaka, K.: Sanitizable Signature with Secret Information. Tokyo Institute of Technology Research Report, C-215, pp. 1–20 (2005)
17. Zhang, F., Safavi-Naini, R., Susilo, W.: ID-Based Chameleon Hashes from Bilinear Pairings. Cryptology ePrint Archive, Report, 2003/208 (2003)