Trapdoor Sanitizable Signatures Made Easy

Dae Hyun Yum, Jae Woo Seo, and Pil Joong Lee

Electronic and Electrical Engineering, POSTECH, Republic of Korea
{dhyum,jwseo,pjl}@postech.ac.kr

Abstract. A sanitizable signature scheme allows a signer to partially delegate signing rights on a message to another party, called a sanitizer. After the message is signed, the sanitizer can modify pre-determined parts of the message and generate a new signature on the sanitized message without interacting with the signer. At ACNS 2008, Canard et al. introduced trapdoor sanitizable signatures based on identity-based chameleon hashes, where the power of sanitization for a given signed message can be delegated to possibly several entities, by giving a trapdoor issued by the signer at any time. We present a generic construction of trapdoor sanitizable signatures from ordinary signature schemes. The construction is intuitively simple and answers the basic theoretic question about the minimal computational complexity assumption under which a trapdoor sanitizable signature exists; one-way functions imply trapdoor sanitizable signatures.

1 Introduction

Message origin authentication can be achieved by digital signatures, since any alteration on signed messages nullifies the validity of digital signatures. However, some applications (e.g., releasing medical documents or classified government documents) require that some parts of a signed document be sanitized without invaliding the signature. This is often called the digital document sanitizing problem and many signature schemes for this problem have been proposed with various names: content extraction signature [1], redactable signature [2], and sanitizable signature [3]. Informally, we will use the term of sanitizable signature to refer to any signature scheme for the digital document sanitizing problem.

A sanitizable signature scheme allows a signer to delegate signing rights on a message to a sanitizer (or a censor) in a limited and controlled way. When generating a signature on a message, the signer selects a specific sanitizer who can later sanitize pre-determined parts of the message and update the signature without interacting with the signer. As pointed out by Ateniese et al. [3], there are many situations where re-signing by the original signer is undesirable, including: (1) the signer's key has expired, (2) the original signature was securely time-stamped, (3) the signer may not be reachable or available, and (4) each new signature would cost too much, either in terms of real expense or in terms of computation. In the literature, there are two different definitions of sanitizable signatures according to the degree of sanitizing capability. The first type is a

J. Zhou and M. Yung (Eds.): ACNS 2010, LNCS 6123, pp. 53–68, 2010.
© Springer-Verlag Berlin Heidelberg 2010

sanitizable signature with pre-determined deletion that allows the sanitizer only to delete pre-determined parts of a signed message (e.g., [1,2,4,5,6]). The second type is a sanitizable signature with pre-determined modification that allows the sanitizer to modify pre-determined parts of the message and generate a new signature on the modified message (e.g., [3,7,8]). The capability of modification is valuable for many applications; especially for authenticated multicast, database outsourcing, and secure routing.

A simple sanitizable signature scheme can be constructed by applying a cryptographic hash function $\mathsf{hash}(\cdot)$ to the concatenation of each submessage with a random number [1]. To generate a signature on a message $m = m_1 \| m_2$ consisting of two submessages m_1 and m_2, the signer computes $\sigma = \mathcal{S}(h_1 \| h_2)$ for $h_i = \mathsf{hash}(m_i \| r_i)$ for $i = 1, 2$, where $\mathcal{S}(\cdot)$ is a signing algorithm of an ordinary signature scheme and r_i is a random number. The sanitizable signature on $m = m_1 \| m_2$ is (σ, r_1, r_2). If a sanitizer wants to delete m_1, he removes m_1 from m and replaces the signature with (σ, h_1, r_2). While verifies can check the validity of the sanitized version of the signed message, they cannot extract any information on the submessage m_1 from h_1 without knowing r_1. In this scheme, either (m_i, r_i) or h_i should be disclosed for verifiers to check the validity of (m, σ) and hence, an undeleted submessage can be sanitized by anyone.

A more sophisticated sanitizable signature scheme was introduced by Ateniese et al. [3], substituting a chameleon hash function $\mathsf{chash}(\cdot)$ for the hash function $\mathsf{hash}(\cdot)$. A chameleon hash [9,10] computed over a submessage m_i with randomness r_i, and under a public key pk is denoted by $\mathsf{chash}_{pk}(m_i, r_i)$. A chameleon hash function (or trapdoor commitment) has the same properties of any cryptographic hash function and, in particular, it provides collision resistance. However, the owner of the secret key sk corresponding to the public key pk can find collisions, i.e., m_i' and r_i' such that $h_i = \mathsf{chash}_{pk}(m_i, r_i) = \mathsf{chash}_{pk}(m_i', r_i')$. For given (m_i, r_i, m_i'), the unique randomness r_i' such that $\mathsf{chash}_{pk}(m_i, r_i) = \mathsf{chash}_{pk}(m_i', r_i')$ can be computed with the knowledge of sk. Chameleon hash functions are always probabilistic algorithms because of the randomness r_i in the input. For a given m_i, there are many hash values $h_i = \mathsf{chash}_{pk}(m_i, r_i)$ by using different randomness r_i. To verify the correctness of a computed chameleon hash value h_i, it is necessary to provide both the submessage m_i and the randomness r_i. If each sanitizer publishes a public key of the chameleon hash function, the signer can choose a specific sanitizer by using the sanitizer's public key. Only the sanitizer who knows the corresponding secret key can modify a signed message by finding collisions of the chameleon hash with a modified message. Several extensions of this approach was presented in [7].

While previous sanitizable signatures divide a message into submessages and then apply a cryptographic primitive (e.g., a hash function or a chameleon hash function) to each submessage, a new methodology utilizing a label was introduced in [11]. By marking the position of sanitizable submessages in a label, a sanitizable signature with constant length can be obtained.

At ACNS 2008, Canard et al. [12] introduced trapdoor sanitizable signatures, where the signer allows a specific user to modify pre-determined parts of a signed

message by producing a piece of information (or trapdoor) that will help the user in sanitizing the message. The signer can choose to whom and when he will deliver the trapdoor information and hence, any user can act as a sanitizer; in other words, a sanitizer does not need any setup procedure such as generating a key pair. To implement the "trapdoor" requirement, Canard et al. [12] gave a construction based on the identity-based chameleon hash function [13]. Instead of the public key pk of chameleon hash functions, identity-based chameleon hash functions can use "identity strings" that are essentially any form of strings such as an e-mail address, a URL, a person's address, and any other unambiguous reference. A secret key sk_{ID} corresponding to an identity string ID is generated by an authority who knows a master secret key msk. To build a trapdoor sanitizable signature scheme, the signer keeps the master secret key msk of the identity-based chameleon hash function. The trapdoor information for sanitizing a message m is essentially the secret key sk_m with respect to the "identity string" m. Consequently, the signer can compute the trapdoor information at any time and give it to whomever he wants.

Our Contribution. We present a very simple and efficient approach to building trapdoor sanitizable signature schemes from ordinary signature schemes. Previous construction of trapdoor sanitizable signature [12] is based on the identity-based chameleon hash function (e.g., [13,14]), which, in turn, is based on the RSA signature or the bilinear maps. The identity-based chameleon hash function is evaluated for each sanitizable submessage. In contrast, our approach requires only one additional signature generation. When instantiated with an aggregate signature scheme (e.g., [15]), the trapdoor sanitizable signature consists of two group elements. The proposed construction answers the basic theoretic question about the minimal computational complexity assumption under which a trapdoor sanitizable signature exists; one-way functions imply trapdoor sanitizable signatures. If we borrow the terminology of [16], this shows that trapdoor sanitizable signature belongs to the "private cryptography world."

2 Preliminaries

If x is a string, then $|x|$ denotes its length, while if X is a finite set, $|X|$ denotes its size. If x and y are strings, $x\|y$ denotes the concatenation of x and y. If k is a positive integer, then $[1,k] = \{1,2,\ldots,k\}$. If $k \in \mathbb{N}$, then 1^k denotes the string of k ones. A function $f(k)$ is negligible if for all polynomial $p(k)$, $f(k) < 1/p(k)$ holds for all sufficiently large $k \in \mathbb{N}$. For a probability space P, $x \leftarrow P$ denotes the algorithm that samples a random element according to P. For a finite set X, $x \leftarrow X$ denotes the algorithm that samples an element uniformly at random from X. If $\mathcal{A}$ is a probabilistic polynomial time (PPT) algorithm, then $z \leftarrow \mathcal{A}(x,y,\ldots)$ denotes the operation of running $\mathcal{A}$ on inputs $x,y,\ldots$ and letting z be the output. If $p(\cdot,\cdot,\cdots)$ is a boolean function, then $\Pr[p(x_1,x_2,\cdots) \mid x_1 \leftarrow P_1,\ x_2 \leftarrow P_2,\ \cdots]$ denotes the probability that $p(x_1,x_2,\cdots)$ is true after executing algorithms $x_1 \leftarrow P_1, x_2 \leftarrow P_2,\cdots$.

Definition 1 (One-way Function). A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is a one-way function, if there exists a polynomial time algorithm which computes $f(x)$ correctly for all x and the following probability is negligible in k for all PPT algorithm $\mathcal{A}$:

$$\Pr[f(x') = y \mid x \leftarrow \{0,1\}^k,\ y = f(x),\ x' \leftarrow \mathcal{A}(y, 1^k)] \qquad \square$$

The existence of one-way functions is the most fundamental assumption of the complexity-based cryptography [17]. The definition of ordinary signature schemes is as follows.

Definition 2 (Digital Signature). A digital signature scheme $\mathcal{DS}$ is a 3-tuple of PPT algorithms $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ that, respectively, generate keys for a user, sign a message, and verify the signature for a message.

- $\mathcal{K}$, the key generation algorithm, is a probabilistic algorithm that takes as input a security parameter and outputs a signing key and a verification key; $(SK, VK) \leftarrow \mathcal{K}(1^k)$.
- $\mathcal{S}$, the signing algorithm, is a probabilistic algorithm that takes as input a message $m \in \{0,1\}^*$ and a signing key SK. It outputs a signature or a special character $\perp$ indicating an error; $\sigma \leftarrow \mathcal{S}(m, SK)$.
- $\mathcal{V}$, the verification algorithm, is a deterministic algorithm that takes as input a message m, a signature σ, and a verification key VK. It outputs a single bit b indicating validity ($b = 1$) or invalidity ($b = 0$) of the signature; $b \leftarrow \mathcal{V}(m, \sigma, VK)$ where $b \in \{0,1\}$. $\square$

We consider existential unforgeability under adaptive chosen message attacks, denoted by UF-CMA [18]. Let $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a signature scheme. The adversary $\mathcal{A}$ is given oracle access to the signing oracle $O^{\mathcal{S}}_{SK}(\cdot)$, i.e., $\mathcal{A}$ is allowed to query the signing oracle $O^{\mathcal{S}}_{SK}(\cdot)$ to obtain valid signatures $\sigma_1, \ldots, \sigma_\lambda$ of arbitrary messages $m_1, \ldots, m_\lambda$ adaptively chosen by $\mathcal{A}$. The adversary succeeds if it forges a valid signature σ of a message m that was not queried to $O^{\mathcal{S}}_{SK}(\cdot)$, i.e., $m \notin \{m_1, \ldots, m_\lambda\}$. An adversary $\mathcal{A}$ is said to (t, λ, ϵ)-break $\mathcal{DS}$, if $\mathcal{A}$ runs in time at most t, makes at most λ queries to the signing oracle, and succeeds in forgery with probability at least ϵ. The signature scheme $\mathcal{DS}$ is said to be (t, λ, ϵ)-secure, if no adversary can (t, λ, ϵ)-break it.

Definition 3 (Secure Signature). The adversary $\mathcal{A}$'s advantage against a signature scheme $\mathcal{DS}$ is defined by

$$\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\mathcal{A},\mathcal{DS}}(k) = \Pr[\mathcal{V}(m, \sigma, VK) = 1 \mid (SK, VK) \leftarrow \mathcal{K}(1^k),\ (m, \sigma) \leftarrow \mathcal{A}^{O^{\mathcal{S}}_{SK}}(VK)]$$

where m should not be queried to the signing oracle. A signature scheme $\mathcal{DS}$ is secure if the advantage of any PPT adversary $\mathcal{A}$ is negligible in k. $\square$

It is well known that a secure signature scheme can be built from a one-way function.

Theorem 1 ([19,20]). *Secure signatures exist if and only if one-way functions exist.* □

Boneh et al. [15] introduced an aggregate signature that allows incremental aggregation of signatures generated by multiple signers into one short signature based on bilinear groups. Let $\mathbb{G}$ and $\mathbb{G}_T$ be multiplicative cyclic groups of prime order p and let g be a generater of $\mathbb{G}$. A bilinear map is a computable map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with the following properties:

- Bilinear: for all $g_1, g_2 \in \mathbb{G}$ and $a, b \in \mathbb{Z}$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
- Non-degenerate: $e(g, g) \neq 1$.

A bilinear group is any group that possesses such a map e and on which CDH (Computational Diffie-Hellman) problem is hard; for given $g, g^a, h \in \mathbb{G}$, it is hard to compute $h^a \in \mathbb{G}$. Let $\mathsf{hash} : \mathbb{G} \times \{0,1\}^* \to \mathbb{G}$ be a cryptographic hash function. The bilinear aggregate signature is defined as follows [21].

Key Generation. For a particular user, pick random $x \leftarrow \mathbb{Z}_p$ and compute $v = g^x$. The user's signing key is (x, v) and the verification key is v.

Signing. For a particular user, given a message $m \in \{0,1\}^*$ and the signing key (x, v), compute $h = \mathsf{hash}(v \| m)$ and $\sigma = h^x$. The signature is $\sigma \in \mathbb{G}$.

Verification. Given a user's verification key v, a message m, and a signature σ, compute $h = \mathsf{hash}(v \| m)$; accept if $e(\sigma, g) = e(h, v)$.

Aggregation. Arbitrarily assign to each user whose signature will be aggregated an index i, ranging from 1 to λ. Each user i provides a signature $\sigma_i \in \mathbb{G}$ on a message $m_i \in \{0,1\}^*$. Compute $\sigma = \prod_{i=t}^{\lambda} \sigma_i$. The aggregate signature is $\sigma \in \mathbb{G}$.

Aggregation Verification. We are given an aggregate signature $\sigma \in \mathbb{G}$ for a set of users, indexed as before, and are given the messages $m_i \in \{0,1\}^*$ and verification keys $v_i \in \mathbb{G}$. To verify the aggregate signature σ, compute $h_i = \mathsf{hash}(v_i \| m_i)$ for $1 \leq i \leq \lambda$, and accept if $e(\sigma, g) = \prod_{i=1}^{\lambda} e(h_i, v_i)$ holds.

3 Trapdoor Sanitizable Signatures

3.1 Definition

A message m is composed of L submessages i.e., $m = m_1 \| \ldots \| m_L$. We assume that each submessage (implicitly or explicitly) includes an index to check that m_i is the i-th submessage; otherwise, we can simply insert an index (e.g., $m_i = i \| m_i$). Let $I_{\mathsf{adm}} \subset [1, L]$ be a set of indices of admissible submessages that can be modified by a sanitizer. We say that I_{adm} is consistent with (m, m') for an original message m and a modified message m' if $\{i \in [1, L] \mid m_i \neq m'_i\} \subset I_{\mathsf{adm}}$. To put it another way, I_{adm} is consistent with (m, m') if a modified message m' agrees with the signer's intention.

Definition 4 (Trapdoor Sanitizable Signature). A trapdoor sanitizable signature scheme TSS is a 6-tuple of PPT algorithms (Setup, KeyGen, Sign, $\mathsf{Trapdoor}$, $\mathsf{Sanitize}$, Vrfy) that, respectively, generate a public parameter, generate keys for a user, sign a message, generate a trapdoor for sanitization, sanitize a signed message, and verify the signature for a message.

- Setup, the parameter generation algorithm, is a probabilistic algorithm that takes as input a security parameter 1^k for $k \in \mathbb{N}$ and outputs a public parameter; $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$.
- KeyGen, the key generation algorithm, is a probabilistic algorithm that takes as input a public parameter and outputs a secret key and a public key; $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$.
- Sign, the signing algorithm, is a probabilistic algorithm that takes as input a message $m = m_1 \| m_2 \| \cdots \| m_L$ for $m_i \in \{0,1\}^*$ and $L \in \mathbb{N}$, a secret key sk, and a set of indices $I_{\mathsf{adm}} \subset [1, L]$. It outputs a signature or a special character $\perp$ indicating an error; $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}, I_{\mathsf{adm}})$.
- $\mathsf{Trapdoor}$, the trapdoor generation algorithm, is a probabilistic algorithm that takes as input a message m, a signature σ, a secret key sk, and a set of indices I_{adm}. It outputs a trapdoor or a special character $\perp$ indicating an error; $\mathsf{td} \leftarrow \mathsf{Trapdoor}(m, \sigma, \mathsf{sk}, I_{\mathsf{adm}})$.
- $\mathsf{Sanitize}$, the sanitization algorithm, is a probabilistic algorithm that takes as input a message m, a signature σ, a public key pk, a trapdoor td, a set of indices I_{adm}, and a modified message m'. It outputs a new signature on the modified message or a special character $\perp$ indicating an error; $\sigma' = \mathsf{Sanitize}(m, \sigma, \mathsf{pk}, \mathsf{td}, I_{\mathsf{adm}}, m')$.
- Vrfy, the verification algorithm, is a deterministic algorithm that takes as input a message m, a signature σ, a public key pk, and a set of indices I_{adm}. It outputs a single bit b indicating validity ($b = 1$) or invalidity ($b = 0$) of the signature; $b \leftarrow \mathsf{Vrfy}(m, \sigma, \mathsf{pk}, I_{\mathsf{adm}})$ where $b \in \{0,1\}$.

The parameter generation algorithm Setup is optional; in this case, Setup is an identity function (i.e., $\mathsf{param} = 1^k$). We omit param in the input of Sign, $\mathsf{Trapdoor}$, $\mathsf{Sanitize}$, and Vrfy, because param can be included in the secret key sk and the public key pk, if necessary. □

Correctness. A trapdoor sanitizable signature scheme should satisfy the standard correctness property of ordinary signature schemes, saying that a genuinely signed message is accepted by the verification algorithm; for any security parameter $k \in \mathbb{N}$, any public parameter $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$, any key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$, any message $m = m_1 \| \cdots \| m_L$ for $L \in \mathbb{N}$, any set of indices $I_{\mathsf{adm}} \subset [1, L]$, and any signature $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}, I_{\mathsf{adm}})$, we have

$$\mathsf{Vrfy}(m, \sigma, \mathsf{pk}, I_{\mathsf{adm}}) = 1.$$

In a trapdoor sanitizable signature scheme, a signature sanitized with a valid trapdoor should also be accepted by the verification algorithm; for any security parameter $k \in \mathbb{N}$, any public parameter $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$, any key pair

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$, any message $m = m_1 \| \cdots \| m_L$ for $L \in \mathbb{N}$, any set of indices $I_{\mathsf{adm}} \subset [1, L]$, any signature $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}, I)$, any trapdoor $\mathsf{td} \leftarrow \mathsf{Trapdoor}(m, \sigma, \mathsf{sk}, I_{\mathsf{adm}})$, and any sanitized signature $\sigma' = \mathsf{Sanitize}(m, \sigma, \mathsf{pk}, \mathsf{td}, I_{\mathsf{adm}}, m')$ on a modified message $m' = m'_1 \| \cdots \| m'_L$ where $\{i \in [1, L] \mid m_i \neq m'_i\} \subset I_{\mathsf{adm}}$, we have

$$\mathsf{Vrfy}(m', \sigma', \mathsf{pk}, I_{\mathsf{adm}}) = 1.$$

Moreover, the sanitization process can be repeated; for any (original or sanitized) signature σ on a (original or sanitized) message m with $\mathsf{Vrfy}(m, \sigma, \mathsf{pk}, I_{\mathsf{adm}}) = 1$, a sanitized signature $\sigma' = \mathsf{Sanitize}(m, \sigma, \mathsf{pk}, \mathsf{td}, I_{\mathsf{adm}}, m')$ on a modified message $m' = m'_1 \| \cdots \| m'_L$ where $\{i \in [1, L] \mid m_i \neq m'_i\} \subset I_{\mathsf{adm}}$ is valid;

$$\mathsf{Vrfy}(m', \sigma', \mathsf{pk}, I_{\mathsf{adm}}) = 1.$$

3.2 Security Model

Ateniese et al. [3] informally introduced five security requirements of sanitizable signature schemes.

- Unforgeability. No outsider should be able to forger the signer's or the sanitizer's signature.
- Immutability. The sanitizer should not be able to modify any part of the message that is not designated as sanitizable by the original signer.
- Transparency. Given a signed message with a valid signature, it should be infeasible to decide whether the message has been sanitized or not.
- Privacy. Sanitized messages and their signatures should not reveal the original data (i.e., the parts which have been sanitized).
- Accountability. In case of a dispute, the signer can prove to a trusted third party (e.g., court) that a certain message was modified by the sanitizer.

Note that unforgeability considers security against outsiders, while immutability against insiders (or sanitizers). These five security requirements were rigorously formalized according to game-based approaches by Brzuska et al. [8]. They distinguished between signer-accountability and sanitizer-accountability and investigated the relationship of the security requirements.

When it comes to trapdoor sanitizable signature schemes, Canard et al. [12] formalized two security requirements: unforgeability and indistinguishability. We rephrase the security model of [12] with game-based approaches. We first define the following oracles that are initialized with a key pair $(\mathsf{sk}, \mathsf{pk})$ and a random bit $b \in \{0, 1\}$.

- A signing oracle $O_{\mathsf{sk}}^{\mathsf{Sign}}(\cdot, \cdot)$ takes a message $m = m_1 \| \cdots \| m_L$, and a set of indices $I_{\mathsf{adm}} \subset [1, L]$ as input and outputs $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}, I_{\mathsf{adm}})$.
- A trapdoor generation oracle $O_{\mathsf{sk}}^{\mathsf{Trapdoor}}(\cdot, \cdot, \cdot)$ takes a message m, a signature σ, and a set of indices I_{adm} as input and outputs $\mathsf{td} \leftarrow \mathsf{Trapdoor}(m, \sigma, \mathsf{sk}, I_{\mathsf{adm}})$.
- A sanitization oracle $O_{\mathsf{sk}}^{\mathsf{Sanitize}}(\cdot, \cdot, \cdot, \cdot)$ takes as input a message m, a signature σ, a set of indices I_{adm}, and a modified message m'. It outputs $\sigma' \leftarrow \mathsf{Sanitize}(m, \sigma, \mathsf{pk}, \mathsf{td}, I_{\mathsf{adm}}, m')$, where $\mathsf{td} \leftarrow \mathsf{Trapdoor}(m, \sigma, \mathsf{sk}, I_{\mathsf{adm}})$.

– A sanitization-or-signing oracle $O^{\mathsf{Sanitize/Sign}}_{\mathsf{sk},b}(\cdot,\cdot,\cdot)$ takes as input a message m, a set of indices I_{adm}, and a modified message m', where I_{adm} should be consistent with (m, m'). If $b = 0$, it outputs $\sigma' \leftarrow \mathsf{Sanitize}(m, \sigma, \mathsf{pk}, \mathsf{td}, I_{\mathsf{adm}}, m')$ where $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}, I_{\mathsf{adm}})$ and $\mathsf{td} \leftarrow \mathsf{Trapdoor}(m, \sigma, \mathsf{sk}, I_{\mathsf{adm}})$. If $b = 1$, it outputs $\sigma' \leftarrow \mathsf{Sign}(m', \mathsf{sk}, I_{\mathsf{adm}})$.
– A left-or-right oracle $O^{\mathsf{LoR}}_{\mathsf{sk},b}(\cdot,\cdot,\cdot,\cdot)$ takes as input two messages m_1, m_2 composed of the same number of submessages, a set of indices I_{adm} and a modified message m', where I_{adm} should be consistent with both (m_1, m') and (m_2, m'). It outputs $\sigma' \leftarrow \mathsf{Sanitize}(m_b, \sigma_b, \mathsf{pk}, \mathsf{td}, I_{\mathsf{adm}}, m')$ where $\sigma_b \leftarrow \mathsf{Sign}(m_b, \mathsf{sk}, I_{\mathsf{adm}})$ and $\mathsf{td} \leftarrow \mathsf{Trapdoor}(m_b, \sigma_b, \mathsf{sk}, I_{\mathsf{adm}})$.

In trapdoor sanitizable signature schemes, a signer can choose any user as a sanitizer just by giving the trapdoor information; thus no clear distinction exists between outsiders and insiders. Accordingly, unforgeability of trapdoor sanitizable signature schemes is defined to include the immutability requirement.

Definition 5 (Unforgeability [12]). A trapdoor sanitizable signature scheme TSS is unforgeable if for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ succeeds in the following game is negligible in the security parameter k:

1. A public parameter param and a key pair $(\mathsf{sk}, \mathsf{pk})$ are generated using $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$ and $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$. The public parameter param and the public key pk are given to $\mathcal{A}$.
2. $\mathcal{A}$ is given access to a signing oracle $O^{\mathsf{Sign}}_{\mathsf{sk}}(\cdot,\cdot)$, a trapdoor generation oracle $O^{\mathsf{Trapdoor}}_{\mathsf{sk}}(\cdot,\cdot,\cdot)$, and a sanitization oracle $O^{\mathsf{Sanitize}}_{\mathsf{sk}}(\cdot,\cdot,\cdot,\cdot)$.
3. $\mathcal{A}$ outputs $(m^*, \sigma^*, I^*_{\mathsf{adm}})$ and succeeds if the following conditions hold.
   (a) $\mathsf{Vrfy}(m^*, \sigma^*, \mathsf{pk}, I^*_{\mathsf{adm}}) = 1$.
   (b) (m^*, σ^*) did not come from the signing oracle. In other words, $\mathcal{A}$ never queried $(m^*, \cdot)$ to the signing oracle.
   (c) (m^*, σ^*) did not come from the sanitization oracle. In other words, $\mathcal{A}$ never queried $(\cdot,\cdot,\cdot, m^*)$ to the sanitization oracle.
   (d) (m^*, σ^*) is not linked to a tuple (td, m, σ) from the trapdoor generation oracle. More precisely, for any message m being in the input of $O^{\mathsf{Trapdoor}}_{\mathsf{sk}}$, there is an index $i \notin I_{\mathsf{adm}}$ such that $m^*_i \neq m_i$, where I_{adm} corresponds to (m, σ). □

The indistinguishability requirement of [12] demands that the output distributions of the sanitization algorithm and the signing algorithm should be identical. That is, the following distributions $\mathcal{D}_{\mathsf{Sanitize}}$ and $\mathcal{D}_{\mathsf{Sign}}$ should be statistically indistinguishable for all key pairs $(\mathsf{sk}, \mathsf{pk})$ and messages $m, m' \in \{0,1\}^*$:

$$\mathcal{D}_{\mathsf{Sanitize}} = \{\sigma' \mid \sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}, I_{\mathsf{adm}}),\ \mathsf{td} \leftarrow \mathsf{Trapdoor}(m, \sigma, \mathsf{sk}, I_{\mathsf{adm}}), \\ \sigma' \leftarrow \mathsf{Sanitize}(m, \sigma, \mathsf{pk}, \mathsf{td}, I_{\mathsf{adm}}, m')\}$$
$$\mathcal{D}_{\mathsf{Sign}} = \{\sigma' \mid \sigma' \leftarrow \mathsf{Sign}(m', \mathsf{sk}, I_{\mathsf{adm}})\}$$

As the indistinguishability requirement of [12] considers statistical difference between the output distributions of the sanitization algorithm and the signing

algorithm, the distinguisher is not given any other information (through oracle queries). By contrast, the transparency requirement of [8], which is defined with computational indistinguishability, captures the indistinguishability requirement of [12] with game-based approaches. We give the transparency requirement of trapdoor sanitizable signatures based on the definition of [8]. The adversary can probe the output distributions adaptively through oracle queries and have additional information by accessing the signing oracle, the sanitization oracle, and the trapdoor generation oracle.

Definition 6 (Transparency). A trapdoor sanitizable signature scheme TSS is transparent if the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible in the security parameter k:

1. A public parameter param and a key pair $(\mathsf{sk}, \mathsf{pk})$ are generated using $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$ and $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$. The public parameter param and the public key pk are given to $\mathcal{A}$.
2. A bit b is randomly chosen and the adversary $\mathcal{A}$ is given access to a signing oracle $O_{\mathsf{sk}}^{\mathsf{Sign}}(\cdot,\cdot)$, a trapdoor generation oracle $O_{\mathsf{sk}}^{\mathsf{Trapdoor}}(\cdot,\cdot,\cdot)$, a sanitization oracle $O_{\mathsf{sk}}^{\mathsf{Sanitize}}(\cdot,\cdot,\cdot,\cdot)$, and a sanitization-or-signing oracle $O_{\mathsf{sk},b}^{\mathsf{Sanitize/Sign}}(\cdot,\cdot,\cdot)$.
3. $\mathcal{A}$ outputs a guess b'.

We say that the adversary $\mathcal{A}$ succeeds if $b' = b$, and denote the probability of this event by $\Pr_{\mathcal{A},\mathsf{TSS}}^{\mathsf{trans}}[\mathsf{Succ}]$. The adversary's advantage is defined as $|\Pr_{\mathcal{A},\mathsf{TSS}}^{\mathsf{trans}}[\mathsf{Succ}] - 1/2|$. □

There are two equivalent ways in formalizing privacy for sanitizable signature schemes [12]; one approach follows semantic security of encryption schemes and the other is based on the indistinguishability notion for encryption. We define the privacy requirement of trapdoor sanitizable signature schemes with the more handy approach of the indistinguishability notion.

Definition 7 (Privacy). A trapdoor sanitizable signature scheme TSS is private if the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible in the security parameter k:

1. A public parameter param and a key pair $(\mathsf{sk}, \mathsf{pk})$ are generated using $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$ and $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$. The public parameter param and the public key pk are given to $\mathcal{A}$.
2. A bit b is randomly chosen and the adversary $\mathcal{A}$ is given access to a signing oracle $O_{\mathsf{sk}}^{\mathsf{Sign}}(\cdot,\cdot)$, a trapdoor generation oracle $O_{\mathsf{sk}}^{\mathsf{Trapdoor}}(\cdot,\cdot,\cdot)$, a sanitization oracle $O_{\mathsf{sk}}^{\mathsf{Sanitize}}(\cdot,\cdot,\cdot,\cdot)$, and a left-or-right oracle $O_{\mathsf{sk},b}^{\mathsf{LoR}}(\cdot,\cdot,\cdot,\cdot)$.
3. $\mathcal{A}$ outputs a guess b'.

We say that the adversary $\mathcal{A}$ succeeds if $b' = b$, and denote the probability of this event by $\Pr_{\mathcal{A},\mathsf{TSS}}^{\mathsf{priv}}[\mathsf{Succ}]$. The adversary's advantage is defined as $|\Pr_{\mathcal{A},\mathsf{TSS}}^{\mathsf{priv}}[\mathsf{Succ}] - 1/2|$. □

Brzuska et al. [8] showed that transparency implies privacy in sanitizable signatures. Their proof of implication can be straightforwardly translated into the

case of trapdoor sanitizable signatures and hence, the same implication holds in trapdoor sanitizable signatures.

Theorem 2. *A transparent trapdoor sanitizable signature scheme is private.*

Proof. In a similar manner to [8], we convert an adversary $\mathcal{A}_{\mathsf{priv}}$ against privacy with a non-negligible advantage ϵ into $\mathcal{A}_{\mathsf{trans}}$ against transparency with a non-negligible advantage $\epsilon/2$ as follows.

$\mathcal{A}_{\mathsf{trans}}^{O_{\mathsf{sk}}^{\mathsf{Sign}}(\cdot,\cdot),O_{\mathsf{sk}}^{\mathsf{Trapdoor}}(\cdot,\cdot,\cdot),O_{\mathsf{sk}}^{\mathsf{Sanitize}}(\cdot,\cdot,\cdot),O_{\mathsf{sk},b}^{\mathsf{Sanitize/Sign}}(\cdot,\cdot,\cdot)}(\mathsf{param},\mathsf{pk})$

$b^* \leftarrow \{0,1\}$

$a \leftarrow \mathcal{A}_{\mathsf{priv}}^{O_{\mathsf{sk}}^{\mathsf{Sign}}(\cdot,\cdot),O_{\mathsf{sk}}^{\mathsf{Trapdoor}}(\cdot,\cdot,\cdot),O_{\mathsf{sk}}^{\mathsf{Sanitize}}(\cdot,\cdot,\cdot),O_{\mathsf{sk},b^*}^{\mathsf{LoR}}(\cdot,\cdot,\cdot)}(\mathsf{param},\mathsf{pk})$

where $O_{\mathsf{sk},b^*}^{\mathsf{LoR}}(m_0,m_1,I_{\mathsf{adm}},m') = O_{\mathsf{sk},b}^{\mathsf{Sanitize/Sign}}(m_{b^*},I_{\mathsf{adm}},m')$ for every query on $(m_0,m_1,I_{\mathsf{adm}},m')$ that $\mathcal{A}_{\mathsf{priv}}$ sends to the left-or-right oracle.

Return $b'=0$ if $a=b^*$, otherwise $b'=1$

$\mathcal{A}_{\mathsf{trans}}$ gets a public parameter param and a public key pk as input and has access to $O_{\mathsf{sk}}^{\mathsf{Sign}}(\cdot,\cdot)$, $O_{\mathsf{sk}}^{\mathsf{Trapdoor}}(\cdot,\cdot,\cdot)$, $O_{\mathsf{sk}}^{\mathsf{Sanitize}}(\cdot,\cdot,\cdot,\cdot)$, and $O_{\mathsf{sk},b}^{\mathsf{Sanitize/Sign}}(\cdot,\cdot,\cdot)$. The goal of $\mathcal{A}_{\mathsf{trans}}$ is to guess a random bit b with non-negligible advantage, i.e., to distinguish whether the message has been sanitized or not. $\mathcal{A}_{\mathsf{trans}}$ simulates the attack environment of $\mathcal{A}_{\mathsf{priv}}$ and utilizes $\mathcal{A}_{\mathsf{priv}}$ as a subroutine. For signing, trapdoor generation, and sanitizing queries of $\mathcal{A}_{\mathsf{priv}}$, $\mathcal{A}_{\mathsf{trans}}$ forwards them to its own oracles and hands the answers back to $\mathcal{A}_{\mathsf{priv}}$. For a left-or-right query of $\mathcal{A}_{\mathsf{priv}}$ on $(m_0,m_1,I_{\mathsf{adm}},m')$, $\mathcal{A}_{\mathsf{trans}}$ sends $(m_{b^*},I_{\mathsf{adm}},m')$ to the sanitization-or-signing oracle and hands the answer back to $\mathcal{A}_{\mathsf{priv}}$. Eventually, $\mathcal{A}_{\mathsf{priv}}$ outputs its guess a, and $\mathcal{A}_{\mathsf{trans}}$ outputs $b'=0$ iff $a=b^*$.

Now, we consider the success probability of $\mathcal{A}_{\mathsf{priv}}$. If $b=0$, then the simulation is identical to the actual attack environment of $\mathcal{A}_{\mathsf{priv}}$ with a random bit b^* and we have $\Pr[b'=0|b=0]=\Pr[\mathcal{A}_{\mathsf{priv}}=b^*]\geq\frac{1}{2}+\epsilon$. If $b=1$, then no information on the bit b^* is given to $\mathcal{A}_{\mathsf{priv}}$ and we have $\Pr[b'=1|b=1]=\frac{1}{2}$. Therefore, the advantage Adv of $\mathcal{A}_{\mathsf{trans}}$ is given as follows.

$$\begin{aligned}\mathsf{Adv} &= \left|\Pr[b'=b]-\frac{1}{2}\right| \\ &= \left|\Pr[b=0]\cdot\Pr[b'=0|b=0]+\Pr[b=1]\cdot\Pr[b'=1|b=1]-\frac{1}{2}\right| \\ &\geq \left|\frac{1}{2}\cdot\left(\frac{1}{2}+\epsilon\right)+\frac{1}{2}\cdot\frac{1}{2}-\frac{1}{2}\right| = \frac{\epsilon}{2}\end{aligned}$$

□

Accountability of sanitizable signatures requires that the origin of a (sanitized) signed message should be undeniable [3,8]. To recognize a sanitizer in any meaningful way, the sanitizer must have some secret information. However, there is no such information in trapdoor sanitizable signatures, because any user, just with a trapdoor, can act as a sanitizer. Therefore, accountability cannot be defined in trapdoor sanitizable signatures.

In summary, the security model of trapdoor sanitizable signatures considers unforgeability and transparency because the definition of unforgeability includes immutability, the transparency requirement implies privacy, and the accountability requirement cannot be defined in a meaningful way.

Definition 8 (Security). A trapdoor sanitizable signature scheme TSS is secure if it is unforgeable and transparent. □

4 Generic Construction from Ordinary Signatures

The previous construction of trapdoor sanitizable signature schemes is based on identity-based chameleon hash functions to implement the trapdoor generation functionality [8]. Identity-based chameleon hash functions are not as efficient as cryptographic hash functions such as SHA-1; rather they are a kind of public-key primitive in terms of functionality and efficiency. Currently known identity-based chameleon hash functions (e.g., [13,14]) are based on the RSA signature or the bilinear maps. Consequently, they are not more efficient than ordinary signature schemes. Recall that the purpose of trapdoor is to delegate the power of signing on a message. Our approach to implementing the trapdoor generation functionality is to generate a temporary key pair $(\hat{SK}, \hat{VK})$ of an ordinary signature scheme and use the signing key $\hat{SK}$ as a trapdoor. This is simple and intuitive. We first give a stateful construction that stores $(\hat{SK}, \hat{VK})$ in a list and then discuss how to remove it.

Construction 1 (Stateful Version). Let $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be an ordinary signature scheme.

- Setup. The public parameter generation algorithm is an identity function. It takes a security parameter 1^k as input and sets $\mathsf{param} = 1^k$.
- KeyGen. The key generation algorithm takes $\mathsf{param} = 1^k$ as input and generates a long-term key pair by executing $(SK, VK) \leftarrow \mathcal{K}(1^k)$. The secret key is $\mathsf{sk} = SK$ and the public key is $\mathsf{pk} = VK$.
- Sign. The signing algorithm takes as input a message $m = m_1 \| m_2 \| \cdots \| m_L$, a secret key $\mathsf{sk} = SK$, and a set of indices $I_{\mathsf{adm}} \subset [1, L]$. It generates a temporary key pair $(\hat{SK}, \hat{VK}) \leftarrow \mathcal{K}(1^k)$ and forms $\hat{m}$ from m and I_{adm} as follows.

$$\hat{m} = \hat{VK} \| \hat{m}_1 \| \hat{m}_2 \| \cdots \| \hat{m}_L, \quad \text{where } \hat{m}_i = \begin{cases} \star & \text{if } \ i \in I_{\mathsf{adm}} \\ m_i & \text{if } \ i \notin I_{\mathsf{adm}} \end{cases} \tag{1}$$

  where $\star$ is a special character for space holder. The signing algorithm computes $\sigma_0 \leftarrow \mathcal{S}(\hat{m}, SK)$ and $\sigma_1 \leftarrow \mathcal{S}(\hat{m} \| m, \hat{SK})$. The signature is $\sigma = (\hat{VK}, \sigma_0, \sigma_1)$. The temporary key pair $(\hat{SK}, \hat{VK})$ is kept in a list List for trapdoor generation.
- Trapdoor. The trapdoor generation algorithm takes as input a message m, a signature σ, a secret key sk, and a set of indices I_{adm}. If σ is a valid signature on m and $\hat{VK}$ exists in List, it outputs a trapdoor $\mathsf{td} = \hat{SK}$ from List. Otherwise, it outputs $\perp$.

– Sanitize. The sanitization algorithm takes as input a message m, a signature $\sigma = (\hat{VK}, \sigma_0, \sigma_1)$, a public key $\mathsf{pk} = VK$, a trapdoor $\mathsf{td} = \hat{SK}$, a set of indices I_{adm}, and a modified message m'. If σ is not a valid signature on m, I_{adm} is not consistent with (m, m'), or $\hat{SK}$ is not a valid signing key with respect to $\hat{VK}$, then it outputs $\perp$. Otherwise, it forms $\hat{m}$ according to Equation (1) and computes $\sigma_1' \leftarrow \mathcal{S}(\hat{m}\|m', \hat{SK})$. The sanitized signature is $\sigma' = (\hat{VK}, \sigma_0, \sigma_1')$.
– Vrfy. The verification algorithm takes as input a message m, a signature $\sigma = (\hat{VK}, \sigma_0, \sigma_1)$, a public key $\mathsf{pk} = VK$, and a set of indices I_{adm}. It forms $\hat{m}$ according to Equation (1) and checks the validity of σ_0 and σ_1 by $b_0 \leftarrow \mathcal{V}(\hat{m}, \sigma_0, VK)$ and $b_1 \leftarrow \mathcal{V}(\hat{m}\|m, \sigma_1, \hat{VK})$. The verification algorithm outputs $b = b_0 \wedge b_1$. □

The signing algorithm Sign generates a signature σ_0 that is essentially a kind of "public-key certificate" of the verification key $\hat{VK}$. The content of the certificate (i.e. $\hat{m}$), which is similar to the label of [11], limits the usage of the key $\hat{VK}$; the corresponding signing key can only be used to generate signatures σ_1 with respect to messages m' where I_{adm} is consistent with (m, m'). This is natural because in a sense, a public-key certificate "delegates" a signing power to a user. The temporary key $\hat{VK}$ is also used for a message ID for $\hat{m} = \hat{VK}\|\hat{m}_1\|\cdots\|\hat{m}_L$.

If we adopt an aggregate signature scheme, the values σ_0 and σ_1 in a signature $\sigma = (\hat{VK}, \sigma_0, \sigma_1)$ can be shortened into one value. For example, if we use the bilinear aggregate signature of Section 2, the signature $\sigma = (\hat{VK}, \sigma_0, \sigma_1)$ on a message m can be compressed into $\sigma = (\alpha, \beta) = (\hat{VK}, \sigma_{01}) = (v, h_0^x h_1^{\hat{x}})$ where $(\mathsf{sk}, \mathsf{pk}) = (x, v)$, $(\hat{SK}, \hat{VK}) = (\hat{x}, \hat{v})$, $h_0 = \mathsf{hash}(v\|\hat{m})$, and $h_1 = \mathsf{hash}(\hat{v}\|\hat{m}\|m)$. An updated signature σ' with a modified message m' and a trapdoor $\mathsf{td} = \hat{SK} = \hat{x}$ can be obtained by $\sigma' = (\alpha, \frac{\beta\sigma_1'}{\sigma_1})$ where $\sigma_1 = \mathsf{hash}(\hat{v}\|\hat{m}\|m)^{\hat{x}}$ and $\sigma_1' = \mathsf{hash}(\hat{v}\|\hat{m}\|m')^{\hat{x}}$. Note that sequential aggregate signatures (e.g., [22,23]) can be used in a similar manner, since σ_1 can always be computed after σ_0.

Theorem 3. *Construction 1 is a secure trapdoor sanitizable signature scheme if the underlying signature scheme is secure.*

Proof. Let $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a secure signature scheme and $\mathsf{TSS}_{\mathcal{DS}}$ be a trapdoor sanitizable signature scheme of Construction 1 instantiated with $\mathcal{DS}$. To prove the security of $\mathsf{TSS}_{\mathcal{DS}}$, we have to show that $\mathsf{TSS}_{\mathcal{DS}}$ is unforgeable and transparent.

Unforgeability. To show the unforgeability of $\mathsf{TSS}_{\mathcal{DS}}$, we build an adversary $\mathcal{A}$ against the signature scheme $\mathcal{DS}$ from an adversary $\mathcal{B}$ breaking the unforgeability of $\mathsf{TSS}_{\mathcal{DS}}$. Recall that $\mathcal{A}$ gets VK as input and has access to the signing oracle $O^{\mathcal{S}}_{SK}$, where (SK, VK) is a valid key pair of $\mathcal{DS}$. On the other hand, $\mathcal{B}$ gets $(\mathsf{param}, \mathsf{pk})$ as input and makes q_{sig} queries to the signing oracle $O^{\mathsf{Sign}}_{\mathsf{sk}}$, q_{tr} queries to the trapdoor generation oracle $O^{\mathsf{Trapdoor}}_{\mathsf{sk}}$, and q_{san} queries to the sanitization oracle $O^{\mathsf{Sanitize}}_{\mathsf{sk}}$. Since a query to the trapdoor generation oracle should contain a valid signature, we have $q_{sig} \geq q_{tr}$ by assuming that the adversary does not make two queries for the same trapdoor.

On input VK, the adversary $\mathcal{A}$ begins simulating the attack environment of $\mathcal{B}$. It first sets $\mathsf{param} = 1^k$ where $k = |VK|$ and generates a long-term key pair $(SK_0, VK_0) \leftarrow \mathcal{K}(1^k)$ and q_{sig} temporary key pairs $(SK_i, VK_i) \leftarrow \mathcal{K}(1^k)$ for $i \in [1, q_{sig}]$. It picks a random number $\mu \in [0, q_{sig}]$ and then sets keys as follows for $i \in [1, q_{sig}]$.

$$\mathsf{sk} = \begin{cases} SK_0 & \text{if } \mu \neq 0 \\ \perp & \text{if } \mu = 0 \end{cases} \qquad \hat{SK}_i = \begin{cases} SK_i & \text{if } \mu \neq i \\ \perp & \text{if } \mu = i \end{cases}$$

$$\mathsf{pk} = \begin{cases} VK_0 & \text{if } \mu \neq 0 \\ VK & \text{if } \mu = 0 \end{cases} \qquad \hat{VK}_i = \begin{cases} VK_i & \text{if } \mu \neq i \\ VK & \text{if } \mu = i \end{cases}$$

The random number μ is $\mathcal{A}$'s guess at the signing key for which $\mathcal{B}$ makes a forgery. If $\mathcal{A}$'s guess is correct, then $\mathcal{A}$ can break the security of $\mathcal{DS}$ from $\mathcal{B}$'s forgery. Otherwise, $\mathcal{A}$ aborts simulation without success. The adversary $\mathcal{A}$ stores all key pairs $(\hat{SK}_i, \hat{VK}_i)$ in List, gives $(\mathsf{param}, \mathsf{pk})$ to $\mathcal{B}$ as input, and answers the queries of $\mathcal{B}$ as follows.

- For the i-th signing query (m, I_{adm}), the adversary $\mathcal{A}$ forms $\hat{m}$ according to Equation (1). If $\mu \neq 0$, then $\mathcal{A}$ generates $\sigma_0 \leftarrow \mathcal{S}(\hat{m}, SK_0)$. Otherwise, $\mathcal{A}$ generates σ_0 by making a query $\hat{m}$ to its own signing oracle $O^{\mathcal{S}}_{SK}(\cdot)$. If $\mu \neq i$, then $\mathcal{A}$ generates $\sigma_1 \leftarrow \mathcal{S}(\hat{m} \| m, \hat{SK}_i)$. Otherwise, $\mathcal{A}$ generates σ_1 by making a query $\hat{m} \| m$ to $O^{\mathcal{S}}_{SK}(\cdot)$. The adversary $\mathcal{A}$ returns the signature $\sigma = (\hat{VK}_i, \sigma_0, \sigma_1)$.
- For the i-th trapdoor query $(m, \sigma, I_{\mathsf{adm}})$ where $\sigma = (\hat{VK}, \sigma_0, \sigma_1)$, the adversary $\mathcal{A}$ first checks the validity of σ. If $\mathsf{Vrfy}(m, \sigma, \mathsf{pk}, I_{\mathsf{adm}}) = 0$, then $\mathcal{A}$ returns $\perp$. If $\mathsf{Vrfy}(m, \sigma, \mathsf{pk}, I_{\mathsf{adm}}) = 1$ and $\hat{VK} = VK$, then $\mathcal{A}$ aborts the simulation. Otherwise, $\mathcal{A}$ returns $\hat{SK}$ corresponding to $\hat{VK}$ from List.
- For the i-th sanitization query $(m, \sigma, I_{\mathsf{adm}}, m')$ where $\sigma = (\hat{VK}, \sigma_0, \sigma_1)$, the adversary $\mathcal{A}$ first checks the validity of the query. If $\mathsf{Vrfy}(m, \sigma, \mathsf{pk}, I_{\mathsf{adm}}) = 0$ or I_{adm} is not consistent with (m, m'), then $\mathcal{A}$ returns $\perp$. Otherwise, $\mathcal{A}$ computes a sanitized signature as follows.
  - If $\hat{VK} \neq VK$, then $\mathcal{A}$ searches $\hat{SK}$ corresponding to $\hat{VK}$ from List and generates $\sigma_1' \leftarrow \mathcal{S}(\hat{m} \| m', \hat{SK})$.
  - Otherwise, $\mathcal{A}$ generates σ_1' by making a query $\hat{m} \| m'$ to its own signing oracle $O^{\mathcal{S}}_{SK}(\cdot)$.

  The adversary $\mathcal{A}$ returns a sanitized signature $\sigma' = (\hat{VK}, \sigma_0, \sigma_1')$.

Let $(m^*, \sigma^*, I^*_{\mathsf{adm}})$ be a successful forgery of $\mathcal{B}$, where $\sigma^* = (\hat{VK}^*, \sigma_0^*, \sigma_1^*)$. That is, $\mathsf{Vrfy}(m^*, \sigma^*, \mathsf{pk}, I^*_{\mathsf{adm}}) = 1$.

- If $\mu = 0$ and σ_0^* is a successful forgery with respect to VK, then $\mathcal{A}$ forms $\hat{m}^*$ from m^* and I^*_{adm} according to Equation (1) and returns $(\hat{m}^*, \sigma_0^*)$ as its own output.
- If $\mu = 0$ and σ_0^* is not a successful forgery with respect to VK, then $\mathcal{A}$ aborts.

- If $\mu \neq 0$ and $\hat{VK}^* = VK$, then $\mathcal{A}$ forms $\hat{m}^*$ from m^* and I^*_{adm} according to Equation (1) and returns $(\hat{m}^* \| m^*, \sigma_1^*)$ as its own output.
- Otherwise (i.e., $\mu \neq 0$ and $\hat{VK}^* \neq VK$), then $\mathcal{A}$ aborts.

If $\mathcal{A}$ does not abort (i.e., $\mu \in [0, q_{sig}]$ is a correct guess), then the simulation is perfect. As $q_{sig} \geq q_{tr}$, the event of $\mathcal{A}$'s not aborting happens with probability at least $1/(q_{sig}+1)$. Therefore, if $\mathcal{B}$ succeeds with a probability ϵ, the adversary $\mathcal{A}$ succeeds with probability at least $\epsilon/(q_{sig}+1)$.

TRANSPARENCY. Recall that a sanitization-or-signing oracle $O^{\mathsf{Sanitize/Sign}}_{\mathsf{sk},b}(\cdot,\cdot,\cdot)$ takes as input a message m, a set of indices I_{adm}, and a modified message m', where I_{adm} should be consistent with (m, m'). If $b = 0$, it outputs $\sigma' \leftarrow \mathsf{Sanitize}(m, \sigma, \mathsf{pk}, \mathsf{td}, I_{\mathsf{adm}}, m')$ and otherwise, it outputs $\sigma' \leftarrow \mathsf{Sign}(m', \mathsf{sk}, I_{\mathsf{adm}})$. The adversary's goal is to guess the random bit $b \in \{0,1\}$. For all messages m, m' and their consistent index set I_{adm}, the scheme $\mathsf{TSS}_{\mathcal{DS}}$ generates the signature $\sigma' = (\hat{VK}, \sigma_0, \sigma_1')$ on message m' as follows.

$$\hat{m}' = \hat{VK} \| \hat{m}'_1 \| \hat{m}'_2 \| \cdots \| \hat{m}'_L, \quad \text{where } \hat{m}'_i = \begin{cases} \star & \text{if } i \in I_{\mathsf{adm}} \\ m'_i & \text{if } i \notin I_{\mathsf{adm}} \end{cases},$$

$$\sigma_0 \leftarrow \mathcal{S}(\hat{m}', SK),$$
$$\sigma_1' \leftarrow \mathcal{S}(\hat{m}' \| m', \hat{SK}).$$

This holds whether the signature σ' is given by $\sigma' \leftarrow \mathsf{Sanitize}(m, \sigma, \mathsf{pk}, \mathsf{td}, I_{\mathsf{adm}}, m')$ or $\sigma' \leftarrow \mathsf{Sign}(m', \mathsf{sk}, I_{\mathsf{adm}})$. In the $\mathsf{TSS}_{\mathcal{DS}}$ scheme, the sanitization-or-signing oracle $O^{\mathsf{Sanitize/Sign}}_{\mathsf{sk},b}(\cdot,\cdot,\cdot)$ works exactly in the same way irrespective of the random bit $b = \{0,1\}$. Therefore, any adversary, which is not necessarily a PPT algorithm, cannot guess the random bit b with a probability greater than $1/2$, even with additional queries to the signing oracle, the trapdoor generation oracle, and the sanitization oracle. In other words, TSS is transparent in an information-theoretic sense. □

Our construction gives a simple answer to the basic theoretic question about the minimal computational complexity assumption under which a trapdoor sanitizable signature exists.

Theorem 4. *Secure trapdoor sanitizable signatures exist if and only if one-way functions exist.*

Proof. Theorem 1 and Theorem 3 proves that one-way functions imply secure trapdoor sanitizable signatures. For the other direction, we use a well-known technique of [20]. Let $\mathsf{TSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Trapdoor}, \mathsf{Sanitize}, \mathsf{Vrfy})$ be a secure trapdoor sanitizable signature. We build a one-way function $f(\cdot)$ as follows; let $f(x)$ for $x \in \{0,1\}^k$ run $\mathsf{param} \leftarrow \mathsf{Setup}(x)$ and $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ and output pk. Assume that there is an adversary $\mathcal{A}$ that can invert $f(\cdot)$. Then, given a public key pk, we would be able to obtain a secret key sk' with the property that sk' could generate signatures valid for pk. But this implies that TSS is insecure, which is a contradiction. □

Gertner et al. [16] minted the terms of "private cryptography world" and "public cryptography world" to divide cryptographic primitives into two groups. The former world consists of one-way function and all its equivalent primitives (e.g., pseudo-random generator, pseudo-random function, bit commitment, and digital signature) and the latter world consists of "harder" primitives such as key agreement, public key encryption, oblivious transfer, secure function evaluation, and trapdoor permutation. According to the terminology of Gertner et al. [16], Theorem 4 shows that trapdoor sanitizable signature belongs to the private cryptography world.

Stateless Version. There is a simple technique to remove the necessity of List in Construction 1. One can make $\mathcal{K}(\cdot)$, the key generation algorithm for temporary key pairs $(\hat{SK}, \hat{VK})$, deterministic. Let $\ell_1(\cdot)$ and $\ell_2(\cdot)$ be polynomials. Let $\mathsf{PRF}_k : \{0,1\}^{\ell_1(k)} \times \{0,1\}^* \to \{0,1\}^{\ell_2(k)}$ be a variable input-length pseudo-random function such as variable input-length MACs (message authentication codes). Let γ_k be the randomness used by $\mathcal{K}(1^k)$ where $|\gamma_k| = \ell_2(k)$. The randomness γ_k can be produced in a way that is deterministically dependent on $\hat{m}$. First, a random key $\kappa \in \{0,1\}^{\ell_1(k)}$ for PRF_k is generated and included in the secret key sk. To sign a message m, the randomness γ_k is computed by $\gamma_k = \mathsf{PRF}_k(\kappa, \hat{m})$. Then, the trapdoor (i.e., the temporary signing key $\hat{SK}$) can be re-computed from κ and $\hat{m}$ without consulting List. No PPT algorithm can distinguish between the stateless version and the stateful version because PRF_k is a pseudorandom function.

References

1. Steinfeld, R., Bull, L., Zheng, Y.: Content extraction signatures. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 285–304. Springer, Heidelberg (2002)
2. Johnson, R., Molnar, D., Song, D.X., Wagner, D.: Homomorphic signature schemes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 244–262. Springer, Heidelberg (2002)
3. 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)
4. 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)
5. Miyazaki, K., Iwamura, M., Matsumoto, T., Sasaki, R., Yoshiura, H., Tezuka, S., Imai, H.: Digitally signed document sanitizing scheme with disclosure condition control. IEICE Transactions 88-A(1), 239–246 (2005)
6. Miyazaki, K., Hanaoka, G., Imai, H.: Invisibly sanitizable digital signature scheme. IEICE Transactions 91-A(1), 392–402 (2008)
7. 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)
8. Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of sanitizable signatures revisited. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 317–336. Springer, Heidelberg (2009)

9. Krawczyk, H., Rabin, T.: Chameleon signatures. In: NDSS. The Internet Society (2000)
10. 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)
11. Yum, D.H., Lee, P.J.: Sanitizable signatures reconsidered (2010) (unpublished manuscript)
12. Canard, S., Laguillaumie, F., Milhau, M.: Trapdoor sanitizable signatures and their application to content protection. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 258–276. Springer, Heidelberg (2008)
13. Ateniese, G., de Medeiros, B.: Identity-based chameleon hash and applications. In: Juels, A. (ed.) FC 2004. LNCS, vol. 3110, pp. 164–180. Springer, Heidelberg (2004)
14. Zhang, F., Safavi-Naini, R., Susilo, W.: ID-based chameleon hashes from bilinear pairings. Cryptology ePrint Archive, Report 2003/208 (2003), http://eprint.iacr.org/
15. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)
16. Gertner, Y., Kannan, S., Malkin, T., Reingold, O., Viswanathan, M.: The relationship between public key encryption and oblivious transfer. In: The 41st Annual Symposium on Foundations of Computer Science, pp. 325–335. IEEE, Los Alamitos (2000)
17. Impagliazzo, R., Luby, M.: One-way functions are essential for complexity based cryptography (extended abstract). In: The 30th Annual Symposium on Foundations of Computer Science, pp. 230–235. IEEE Computer Society, Los Alamitos (1989)
18. 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)
19. Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In: The 21st Annual ACM Symposium on Theory of Computing, pp. 33–43. ACM, New York (1989)
20. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: The 22nd Annual ACM Symposium on Theory of Computing, pp. 387–394. ACM, New York (1990)
21. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: A survey of two signature aggregation techniques. CryptoBytes 6(2), 1–10 (2003)
22. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (2004)
23. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 465–485. Springer, Heidelberg (2006)