2013

Accountable Trapdoor Sanitizable Signatures

Junzuo LAI

Xuhua DING
Singapore Management University, xhding@smu.edu.sg

Yongdong Wu

Citation

LAI, Junzuo; DING, Xuhua; and Wu, Yongdong. Accountable Trapdoor Sanitizable Signatures. (2013). *Information Security Practice and Experience: 9th International Conference, ISPEC 2013, Lanzhou, China, May 12-14, 2013: Proceedings*. 7863, 117-131.
Available at: https://ink.library.smu.edu.sg/sis_research/1971

Accountable Trapdoor Sanitizable Signatures

No Author Given

No Institute Given

Abstract. Sanitizable signature (SS) allows a signer to partly delegate signing rights to *a predetermined* party, called sanitizer, who can later modify certain designated parts of a message originally signed by the signer and generate a new signature on the sanitized message without interacting with the signer. One of the important security requirements of sanitizable signatures is *accountability*, which allows the signer to prove, in case of dispute, to a third party that a message was modified by the sanitizer. Trapdoor sanitizable signature (TSS) enables a signer of a message to delegate the power of sanitization to *any parties at anytime* but at the expense of losing the accountability property. In this paper, we introduce the notion of accountable trapdoor sanitizable signature (ATSS) which lies between SS and TSS. As a building block for constructing ATSS, we also introduce the notion of accountable chameleon hash (ACH), which is an extension of chameleon hash (CH) and might be of independent interest. We propose a concrete construction of ACH and show how to use it to construct an ATSS scheme.

Key words: Trapdoor Sanitizable Signature, Accountability, Chameleon Hash

1 Introduction

Ateniese et al. [1] introduced the notion of sanitizable signature (SS) and presented a generic construction based on chameleon hash (CH) [18]. Sanitizable signatures allow a signer to partly delegate signing rights to a predetermined party, called a sanitizer. During signature generation on a message, the signer chooses a specific sanitizer who, with the knowledge of the putative signature, can later modify certain designated parts of the message and generate a new signature on the sanitized message without interacting with the signer. The capability of modification renders sanitizable signatures useful in many applications, such as authenticated multicast, authenticated database outsourcing and authenticated multimedia content distribution.

Sanitizable signatures are required to possess the following five security properties [1]:

- Unforgeability. An outsider (i.e., neither the signer nor the sanitizer) should not be able to forge the signer's or the sanitizer's signature.
- Immutability. The sanitizer should not be able to produce valid signatures for messages where it has modified other than the designated parts.
- Privacy. Sanitized messages and their signatures should not reveal the original data.
- Transparency. An outsider should not be able to decide whether a message has been sanitized or not.
- Accountability. In case of a dispute, the signer can prove to a trusted third party that a certain message was modified by the sanitizer.

Subsequently Canard et al. [8] introduced the notion of trapdoor sanitizable signature (TSS), which is an extension of SS. TSS enables a signer to delegate the power of sanitization for a signed message to any party. They also proposed a generic construction of TSS based on identity-based chameleon hash (IBCH) [2]. However, TSS does not satisfy the security requirement of accountability.

In this paper, we introduce the notion of accountable TSS (ATSS) which lies between TSS and SS. Like SS, our ATSS scheme satisfies the accountability property; it allows a signer to generate an ATSS signature on a message for a predetermined candidate sanitizer. The ATSS signature alone does not provide the candidate sanitizer the power to modify the underlying message and generate a valid signature on the modified message. In order to generate a valid signature on a modified message, similar to TSS, the candidate sanitizer needs to obtain a trapdoor from the signer, in addition to the signer's signature.

One possible application of ATSS is content authentication in tiered multimedia distribution systems [12]. Such a system consists of a top-tier primary content provider and a number of lower-tier affiliating providers each with its own users. An example is a movie distributer in Hollywood that has a number of country distributers worldwide. Whenever a new movie is released, the Hollywood distributer signs the video stream for a country distributor using ATSS and sends the video stream and the signature to the country distributor for previewing. Upon receiving the video stream, the country distributor verifies the authenticity of the movie using the signature. If the country distributor is interested in distributing the movie, it enters a contract with or make a payment to the Hollywood distributor. The latter in turn sends a trapdoor to the former. With the knowledge of the trapdoor, the country distributor can then modify/adapt the movie for its local market (e. g., adding subtitles in the local language) and generate a valid signature on the modified video stream.

1.1 Our Contribution

Contributions of the paper can be summarized as follows:

1. We introduce the notion of ATSS. In an ATSS scheme, a signer needs to predetermine a user as a candidate sanitizer during the signature generation. This signature alone does not allow the candidate sanitizer to produce a new signature on a sanitized message. To generate a new signature on a sanitized message, the candidate sanitizer needs to obtain a trapdoor from the original signer.
2. We extend the notion of CH by introducing the notion of accountable CH (ACH) and define its security requirements. We propose a concrete construction of ACH that satisfies the security requirements in the random oracle model [6].
3. Based on ACH, we present a generic construction of ATSS. Instantiating the generic construction with our concrete ACH scheme, we can obtain the first ATSS scheme.

1.2 Related Work

Sanitizable Signature. The notion of SS was introduced by Ateniese et al. [1]. Such signatures allow a sanitizer to modify certain designated parts of a signed message and generate a new signature on the sanitized message without interacting with the signer. Klonowski and Lauks [17] presented several extensions of SS, including limitation of the set of possible modifications of a single mutable block and limitation of the number of modifications of mutable blocks. Pöhls et al. [22] integrated SS schemes into the XML signature specification.

Ateniese et al. [1] identified five security requirements of SS schemes, *unforgeability*, *immutability*, *privacy*, *transparency* and *accountability*. Brzuska et al. [7] revisited these security requirements and investigated their relationships, showing for example that transparency implies privacy.

Miyazaki et al. [21] used the notion of SS in a slightly different vein. Their SS schemes [21, 15, 20] allow a sanitizer to only *delete* predetermined parts of a signed message.

The notion of incremental cryptography [5] and homomorphic signatures, which encompass transitive [19], redactable [16] and context-extraction signatures [24], are also related to SS. We refer the reader to [1] for details.

Trapdoor Sanitizable Signature. Canard et al. [8] introduced the notion of TSS, in which the power of sanitization is given to possibly several entities. Based on IBCH, Canard et al. [8] proposed a generic construction of TSS. Recently, Yum et al. [25] presented another generic construction of TSS from ordinary signature schemes; therefore, one-way functions imply TSS. Bao et al. [4] extended TSS for the hierarchical setting.

Chameleon Hash. Our work is also related to CH functions, which are randomized collision-resistant hash functions with the additional property that given a trapdoor, one can efficiently generate collisions. CH was first introduced by Krawczyk and Rabin [18]. Other CH constructions were proposed subsequently [9, 3, 14, 13, 11] .

Ateniese and Medeiros [2] extended CH to identity-based setting [23] and introduced the notion of IBCH. Zhang et al. [26] and Chen et al. [10] followed their work.

1.3 Organization

The rest of the paper is organized as follows. Some preliminaries are given in Section 2. The notion and security requirements of ATSS are introduced in Section 3. In Section 4, we present the notion and security requirements of ACH, and propose a concrete construction. In Section 5, we propose a generic construction of ATSS from ACH and present a specific ATSS scheme based on our ACH scheme. Finally, we state our conclusion in Section 6.

2 Preliminaries

If L is a positive integer, then $[1, L] = \{1, 2, \ldots, L\}$. If A, B are two sets, $A \backslash B = \{x \in A | x \notin B\}$. If $x_1, x_2, \ldots$ are strings, then $x_1 \| x_2 \| \ldots$ denotes their concatenation. We denote by $\mathcal{R}$ the range of random number. We say that a function $f(\lambda)$ is *negligible* if for every $c > 0$ there exists an λ_c such that $f(\lambda) < 1/\lambda^c$ for all $\lambda > \lambda_c$.

2.1 Bilinear Pairings

Let $\mathbb{G}$ be a cyclic multiplicative group of prime order p and $\mathbb{G}_T$ be a cyclic multiplicative group of the same order p. A bilinear pairing is a map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with the following properties:

- Bilinearity: $\forall g_1, g_2 \in \mathbb{G}, \forall a, b \in \mathbb{Z}_p^*$, we have $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$;
- Non-degeneracy: There exist $g_1, g_2 \in \mathbb{G}$ such that $e(g_1, g_2) \neq 1$;
- Computability: There exists an efficient algorithm to compute $e(g_1, g_2)$ for $\forall g_1, g_2 \in \mathbb{G}$.

2.2 Computational Diffie-Hellman (CDH) Assumption

The security of our ACH scheme will be reduced to the hardness of the computational Diffie-Hellman (CDH) problem in the bilinear map group system $\langle p, \mathbb{G}, \mathbb{G}_T, e \rangle$ in which the ACH scheme is constructed.

Definition 1 *Given a bilinear map group system $\langle p, \mathbb{G}, \mathbb{G}_T, e \rangle$, a generator g of $\mathbb{G}$ and elements $g^a, g^b \in \mathbb{G}$ where a, b are selected uniformly at random from $\mathbb{Z}_p^*$, the CDH problem in the bilinear map group system is to compute g^{ab}. We say that the CDH assumption holds in a bilinear map group system $\langle p, \mathbb{G}, \mathbb{G}_T, e \rangle$ if no probabilistic polynomial-time algorithm can solve the CDH problem in the bilinear map group system with non-negligible probability.*

3 Accountable Trapdoor Sanitizable Signature and Its Security Requirements

ATSS lies between sanitizable signature and trapdoor sanitizable signature. Like a TSS scheme, an ATSS scheme includes the algorithms: GlobalSetup, KeyGen, Sign, Trapdoor, Sanitize and Verify. However, in ATSS, besides the private key of the signer, the inputs of the Sign algorithm include the public key of a candidate sanitizer and a transaction identifer TID. In order to generate a new signature on a sanitized message, in ATSS, the inputs of the Sanitize algorithm include the private key of the candidate sanitizer and a trapdoor associated with the transaction identifier TID generated by the signer using Trapdoor algorithm, not just a trapdoor associated with the transaction identifier as in TSS or just the private key of the sanitizer as in SS.

In addition, to settle disputes about the origin of a message-signature pair, an algorithm Proof enables the signer to produce a proof π. The proof π is generated from a set of previously signed messages. A Judge algorithm then uses the proof π to decide if a valid message-signature pair was created by the signer or the sanitizer (the lack of such a proof is interpreted as a signer origin).

Concretely, an ATSS scheme is a tuple of algorithms described as follows:

GlobalSetup takes as input a security parameter λ. It produces a common public parameter *param* to be used by all parties in the system.

$$param \leftarrow \mathsf{GlobalSetup}(\lambda).$$

KeyGen takes as input a security parameter λ and the common public parameter $param$. It generates a public/private key pair (pk, sk). Every party in the system uses this randomized algorithm to generate a private/public key pair himself or herself.

$$(pk, sk) \leftarrow \mathsf{KeyGen}(\lambda, param).$$

For presentation simplicity, we assume there exist a single singer and multiple sanitizers in the system. We denote by (pk_{sig}, sk_{sig}) the key pair of the signer, and by (pk_{san}, sk_{san}) the key pair of a sanitizer.

Sign takes as input a sanitizer's public key pk_{san}, a message $m = m_1 \| \cdots \| m_L$, a set of indices $I \subseteq [1, L]$ that are sanitizable, a transaction identifier TID and the signer's private key sk_{sig}. It outputs a signature σ on m.

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

We assume that each message signed has a unique transaction identifier.

$\mathsf{Trapdoor}$ takes as input a message m, a set of indices I that are sanitizable, a transaction identifier TID, a valid signature σ on $(pk_{sig}, pk_{san}, m, I, \mathsf{TID})$ and the signer's private key sk_{sig}. It outputs a trapdoor td_{TID}.

$$td_{\mathsf{TID}} \leftarrow \mathsf{Trapdoor}(m, I, \mathsf{TID}, \sigma, sk_{sig}).$$

$\mathsf{Sanitize}$ takes as input the signer's public key pk_{sig}, a message m, a set of the indices I that are sanitizable, a transaction identifier TID, a valid signature σ on $(pk_{sig}, pk_{san}, m, I, \mathsf{TID})$, a trapdoor td_{TID} associated with TID, the sanitizer's private key sk_{san} and a new message m'. It outputs a new signature σ' on $(pk_{sig}, pk_{san}, m', I, \mathsf{TID})$.

$$\sigma' \leftarrow \mathsf{Sanitize}(pk_{sig}, m, I, \mathsf{TID}, \sigma, m', td_{\mathsf{TID}}, sk_{san}).$$

Verify takes as input $param$, the signer's public key pk_{sig}, a sanitizer's public key pk_{san}, a message m, a set of the indices I that are sanitizable, a transaction identifier TID and a putative signature σ. It outputs 1 if the signature σ on m is valid and 0 otherwise.

$$0/1 \leftarrow \mathsf{Verify}(param, pk_{sig}, pk_{san}, m, I, \mathsf{TID}, \sigma).$$

Proof takes as input $param$, the signer's private key sk_{sig}, a valid signature σ on $(pk_{sig}, pk_{san}, m, I, \mathsf{TID})$, and a set of (polynomially many) additional message-signature pairs $((pk_{sig}, pk_{san}^{(i)}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})_{i=1,2,\ldots,q}$. It outputs a proof $\pi \in \{0,1\}^*$.

$$\pi \leftarrow \mathsf{Proof}(param, sk_{sig}, ((pk_{sig}, pk_{san}, m, I, \mathsf{TID}), \sigma), ((pk_{sig}, pk_{san}^{(i)}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})_{i=1,2,\ldots,q}).$$

Judge takes as input $param$, the signer's public key pk_{sig}, the sanitizer's public key pk_{san}, a valid signature σ on $(pk_{sig}, pk_{san}, m, I, \mathsf{TID})$ and a proof π. It outputs a decision $d \in \{\mathtt{Sig}/\mathtt{San}\}$ indicating whether the message-signature pair $((pk_{sig}, pk_{san}, m, I, \mathsf{TID}), \sigma)$ was created by the signer or the sanitizer.

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

The usual correctness properties should hold for an ATSS scheme, saying that genuinely signed or sanitized messages are accepted. Formally, for any security parameter λ, any message $m = m_1 \| \cdots \| m_L$, any set of indices $I \subseteq [1, L]$, any transaction identifier TID, $param \leftarrow \mathsf{GlobalSetup}(\lambda)$, $(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KeyGen}(\lambda, param)$, $(pk_{san}, sk_{san}) \leftarrow \mathsf{KeyGen}(\lambda, param)$, $\sigma \leftarrow \mathsf{Sign}(pk_{san}, m, I, \mathsf{TID}, sk_{sig})$, $td_{\mathsf{TID}} \leftarrow \mathsf{Trapdoor}(m, I, \mathsf{TID}, \sigma, sk_{sig})$, and $\sigma' \leftarrow \mathsf{Sanitize}(pk_{sig}, m, I, \mathsf{TID}, \sigma, m', td_{\mathsf{TID}}, sk_{san})$, we must have

$$\mathsf{Verify}(param, pk_{sig}, pk_{san}, m, I, \mathsf{TID}, \sigma) = 1 \text{ and } \mathsf{Verify}(param, pk_{sig}, pk_{san}, m', I, \mathsf{TID}, \sigma') = 1.$$

The security requirements of an ATSS scheme include *unforgeability*, *indistinguishability* and *accountability*. The *unforgeability* and *indistinguishability* requirements of ATSS are extended from the security requirements of TSS [8]. Informally, unforgeability requires that an outsider be not able to forge a signature on the original or the sanitized message, and indistinguishability requires that an outsider be not able to decide whether a message has been sanitized or not.

Accountability requires that the origin of a (sanitized) signature be undeniable. We distinguish between *sanitizer-* and *signer-accountability*, as did in [7]. Informally, *sanitizer-accountability* implies that if a message has not been signed by the signer, then even a malicious sanitizer should not be able to make a judge accuse the signer, and *signer-accountability* implies that if a signed message has not been sanitized, then even a malicious signer should not be able to make the judge accuse the sanitizer.

Unforgeability: An ATSS scheme is existentially unforgeable under adaptive chosen message attacks, if for any probabilistic polynomial-time adversary $\mathcal{A}$, the probability that $\mathcal{A}$ succeeds in the following game between $\mathcal{A}$ and a challenger is negligible in the security parameter λ:

Setup The challenger runs

$$param \leftarrow \mathsf{GlobalSetup}(\lambda),\ (pk_{sig}, sk_{sig}) \leftarrow \mathsf{KeyGen}(\lambda, param),\ (pk_{san}, sk_{san}) \leftarrow \mathsf{KeyGen}(\lambda, param),$$

and sends the common public parameter $param$, the signer's public key pk_{sig} and the sanitizer's public key pk_{san} to the adversary $\mathcal{A}$.

Query phase The adversary $\mathcal{A}$ adaptively issues queries:

1. $\mathcal{O}_{ATSS}^{\mathsf{Sign}}$ query on (m, I, TID), where $I \subseteq [1, L]$ is a set of indices and TID is a transaction identifier: The challenger forwards the valid signature σ on $(pk_{sig}, pk_{san}, m, I, \mathsf{TID})$ to the adversary.
2. $\mathcal{O}_{ATSS}^{\mathsf{Trapdoor}}$ query on $(m, I, \mathsf{TID}, \sigma)$, where σ is a valid signature on $(pk_{sig}, pk_{san}, m, I, \mathsf{TID})$: The challenger forwards the corresponding trapdoor td_{TID} to the adversary.
3. $\mathcal{O}_{ATSS}^{\mathsf{Sanitize}}$ query on $(m, I, \mathsf{TID}, \sigma, m')$, where $m_i = m'_i$ for all $i \notin I$: The challenger forwards the new valid signature σ' on $(pk_{sig}, pk_{san}, m', I, \mathsf{TID})$ to the adversary.

Output $\mathcal{A}$ outputs $(m^*, I^*, \mathsf{TID}^*, \sigma^*)$ and succeeds if the following conditions hold.

1. $\mathsf{Verify}(param, pk_{sig}, pk_{san}, m^*, I^*, \mathsf{TID}^*, \sigma^*) = 1$.
2. $\mathcal{A}$ never queries $\mathcal{O}_{ATSS}^{\mathsf{Sign}}$ oracle on $(m^*, I^*, \mathsf{TID}^*)$.
3. (m^*, σ^*) does not come from $\mathcal{O}_{ATSS}^{\mathsf{Sanitize}}$ oracle, i.e., $\mathcal{A}$ never queries $\mathcal{O}_{ATSS}^{\mathsf{Sanitize}}$ oracle on $(m, I^*, \mathsf{TID}^*, \sigma, m^*)$, where σ is a valid signature on $(pk_{sig}, pk_{san}, m, I^*, \mathsf{TID}^*)$ and $m_i = m_i^*$ for all $i \notin I^*$.
4. $\mathcal{A}$ never queries $\mathcal{O}_{ATSS}^{\mathsf{Trapdoor}}$ oracle on $(m, I^*, \mathsf{TID}^*, \sigma)$, where σ is a valid signature on $(pk_{sig}, pk_{san}, m, I^*, \mathsf{TID}^*)$ and $m_i = m_i^*$ for all $i \notin I^*$.

Indistinguishability: The indistinguishability of an ATSS scheme requires that the output distributions of Sign algorithm and $\mathsf{Sanitize}$ algorithm be computational indistinguishable. In other words, for all sufficiently large λ, any $param \leftarrow \mathsf{GlobalSetup}(\lambda)$, $(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KeyGen}(\lambda, param)$, $(pk_{san}, sk_{san}) \leftarrow \mathsf{KeyGen}(\lambda, param)$, any set of indices $I \subseteq [1, L]$, any message pairs m, m' such that $m_i = m'_i$ for all $i \notin I$, any transaction identifier TID, the following distribution ensembles $\mathcal{D}_{\mathsf{Sanitize}}$ and $\mathcal{D}_{\mathsf{Sign}}$ are computational indistinguishable:

$$\begin{aligned}\mathcal{D}_{\mathsf{Sanitize}} &= \{(m', \hat{\sigma}) | \sigma \leftarrow \mathsf{Sign}(pk_{san}, m, I, \mathsf{TID}, sk_{sig}), td_{\mathsf{TID}} \leftarrow \mathsf{Trapdoor}(m, I, \mathsf{TID}, \sigma, sk_{sig}),\\ &\qquad \hat{\sigma} \leftarrow \mathsf{Sanitize}(pk_{sig}, m, I, \mathsf{TID}, \sigma, m', td_{\mathsf{TID}}, sk_{san})\}_{\lambda, param, pk_{sig}, pk_{san}, I, \mathsf{TID}},\\ \mathcal{D}_{\mathsf{Sign}} &= \{(m', \sigma') | \sigma' \leftarrow \mathsf{Sign}(pk_{san}, m', I, \mathsf{TID}, sk_{sig})\}_{\lambda, param, pk_{sig}, pk_{san}, I, \mathsf{TID}}.\end{aligned}$$

Sanitizer-accountability: An ATSS scheme is sanitizer-accountable, if for any probabilistic polynomial-time adversary $\mathcal{A}$, the probability that $\mathcal{A}$ succeeds in the following game between $\mathcal{A}$ and a challenger is negligible in the security parameter λ:

Setup The challenger runs

$$param \leftarrow \mathsf{GlobalSetup}(\lambda),\ (pk_{sig}, sk_{sig}) \leftarrow \mathsf{KeyGen}(\lambda, param),$$

and sends the common public parameter $param$ and the signer's public key pk_{sig} to the adversary $\mathcal{A}$.

Query phase The adversary $\mathcal{A}$ adaptively issues queries:

1. $\mathcal{O}_{ATSS}^{\mathsf{Sign}}$ query on $(pk_{san}, m, I, \mathsf{TID})$, where pk_{san} is a sanitizer's public key chosen by $\mathcal{A}$, $I \subseteq [1, L]$ is a set of indices and TID is a transaction identifier: The challenger forwards the valid signature σ on $(pk_{sig}, pk_{san}, m, I, \mathsf{TID})$ to the adversary.
2. $\mathcal{O}_{ATSS}^{\mathsf{Trapdoor}}$ query on $(pk_{san}, m, I, \mathsf{TID}, \sigma)$, where σ is a valid signature on $(pk_{sig}, pk_{san}, m, I, \mathsf{TID})$: The challenger forwards the corresponding trapdoor td_{TID} to the adversary.

Output $\mathcal{A}$ outputs $(pk_{sig}, pk^*_{san}, m^*, I^*, \mathsf{TID}^*, \sigma^*)$ and succeeds if the following conditions hold.

1. $\mathsf{Verify}(param, pk_{sig}, pk^*_{san}, m^*, I^*, \mathsf{TID}^*, \sigma^*) = 1$.
2. $((pk_{sig}, pk^*_{san}, m^*, I^*, \mathsf{TID}^*), \sigma^*) \neq ((pk_{sig}, pk^{(i)}_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})$ for all $i = 1, 2, \ldots, q$, where $(pk^{(i)}_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)})$ and $\sigma^{(i)}$ for $i = 1, 2, \ldots, q$ denote the queries and answers to and from oracle $\mathcal{O}_{ATSS}^{\mathsf{Sign}}$.
3. $\mathtt{Sig} \leftarrow \mathsf{Judge}(param, ((pk_{sig}, pk^*_{san}, m^*, I^*, \mathsf{TID}^*), \sigma^*), \pi^*)$, where

$$\pi^* \leftarrow \mathsf{Proof}(param, sk_{sig}, ((pk_{sig}, pk^*_{san}, m^*, I^*, \mathsf{TID}^*), \sigma^*), ((pk_{sig}, pk^{(i)}_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})_{i=1,2,\ldots,q}).$$

Signer-accountability: An ATSS scheme is signer-accountable, if for any probabilistic polynomial-time adversary $\mathcal{A}$, the probability that $\mathcal{A}$ succeeds in the following game between $\mathcal{A}$ and a challenger is negligible in the security parameter λ:

Setup The challenger runs

$$param \leftarrow \mathsf{GlobalSetup}(\lambda),\ (pk_{san}, sk_{san}) \leftarrow \mathsf{KeyGen}(\lambda, param),$$

and sends the common public parameters $param$ and the sanitizer's public key pk_{san} to the adversary $\mathcal{A}$.

Query phase The adversary $\mathcal{A}$ adaptively issues $\mathcal{O}_{ATSS}^{\mathsf{Sanitize}}$ query on $(pk_{sig}, m, I, \mathsf{TID}, \sigma, td_{\mathsf{TID}}, m')$, where pk_{sig} is a singer's public key chosen by $\mathcal{A}$, σ is a valid signature on $(pk_{sig}, pk_{san}, m, I, \mathsf{TID})$, td_{TID} is the trapdoor associated with TID and $m_i = m'_i$ for all $i \notin I$: The challenger forwards the new valid signature σ' on $(pk_{sig}, pk_{san}, m', I, \mathsf{TID})$ to the adversary.

Output $\mathcal{A}$ outputs $(pk^*_{sig}, pk_{san}, m^*, I^*, \mathsf{TID}^*, \sigma^*, \pi^*)$ and succeeds if the following conditions hold.

1. $\mathsf{Verify}(param, pk^*_{sig}, pk_{san}, m^*, I^*, \mathsf{TID}^*, \sigma^*) = 1$.
2. $\mathtt{San} \leftarrow \mathsf{Judge}(param, ((pk^*_{sig}, pk_{san}, m^*, I^*, \mathsf{TID}^*), \sigma^*), \pi^*)$.
3. $((pk^*_{sig}, pk_{san}, m^*, I^*, \mathsf{TID}^*), \sigma^*) \neq ((pk^{(i)}_{sig}, pk_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})$ for all $i = 1, 2, \ldots, q$, where $((pk^{(i)}_{sig}, pk_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})$ for $i = 1, 2, \ldots, q$ denote the answers from oracle $\mathcal{O}_{ATSS}^{\mathsf{Sanitize}}$.

4 Accountable Chameleon Hash and Its Construction

In this section, we first introduce and formulate accountable chameleon hash (ACH). Then, we present a construction of ACH and analyze its security in the random oracle model.

4.1 Accountable Chameleon Hash

ACH is a new paradigm which lies between chameleon hash (CH) and identity-based chameleon hash (IBCH). The inputs of the Hash algorithm of an ACH include two users' public keys and a transaction identifier TID, not just the public key of a single user as in a CH scheme or just an identity as in an IBCH scheme. In order to find a collision, the inputs of the Forge algorithm of an ACH scheme include one user's private key and a trapdoor information associated with the transaction identifer TID generated by the other user, not just a single user's private key as in a CH scheme or just a trapdoor information associated with an identity as in an IBCH scheme.

Concretely, an ACH scheme consists of the following algorithms:

$\mathsf{GlobalSetup}$ takes as input a security parameter λ. It produces a common public parameter $param$ to be used by all parties in the system.

$$param \leftarrow \mathsf{GlobalSetup}(\lambda).$$

KeyGen takes as input a security parameter λ and the common public parameter $param$. It generates a public/private key pair (pk, sk). All parties in the system use this randomized algorithm to generate a private/public key pair himself or herself.

$$(pk, sk) \leftarrow \mathsf{KeyGen}(\lambda, param).$$

Hash takes as input $param$, user i's public key pk_i, user j's public key pk_j, a message m and a unique transaction identifier TID. It chooses a random r and outputs a hash value h.

$$h \leftarrow \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m, r).$$

$\mathsf{Trapdoor}$ takes as input user i's private key sk_i and a transaction identifier TID. It outputs the trapdoor information $td_{i,\mathsf{TID}}$ associated with user i and the transaction identifier TID.

$$td_{i,\mathsf{TID}} \leftarrow \mathsf{Trapdoor}(sk_i, \mathsf{TID}).$$

Forge takes as input user j's private key sk_j, the trapdoor information $td_{i,\mathsf{TID}}$ associated with user i and a transaction identifier TID, the hash value h on a message m with user i's public key pk_i, user j's public key pk_j, the transaction identifier TID, random r, and a message m'. It outputs a random r'.

$$r' \leftarrow \mathsf{Forge}(sk_j, td_{i,\mathsf{TID}}, pk_i, pk_j, \mathsf{TID}, m, r, h, m').$$

For correctness, we require that

$$\mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m, r) = h = \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m', r') \text{ and } m' \neq m,$$

where

$$r' \leftarrow \mathsf{Forge}(sk_j, td_{i,\mathsf{TID}}, pk_i, pk_j, \mathsf{TID}, m, r, h, m'),\ td_{i,\mathsf{TID}} \leftarrow \mathsf{Trapdoor}(sk_i, \mathsf{TID}).$$

The security of an ACH scheme consists of two requirements: *resistance to collision forgery under active attacks* and *forgery indistinguishability*.

Resistance to collision forgery under active attacks: The accountable chameleon hash scheme is secure against (existential) collision forgery under active attacks if, for any probabilistic polynomial-time algorithm $\mathcal{A}$, the probability that $\mathcal{A}$ succeeds in the following game between $\mathcal{A}$ and a challenger is negligible in the security parameter λ:

Setup The challenger runs

$$param \leftarrow \mathsf{GlobalSetup}(\lambda),\ (pk_i, sk_i) \leftarrow \mathsf{KeyGen}(\lambda, param),\ (pk_j, sk_j) \leftarrow \mathsf{KeyGen}(\lambda, param),$$

and sends the common public parameter $param$, user i's public key pk_i and user j's public/private key pair (pk_j, sk_j) to the adversary $\mathcal{A}$.

Query phase The adversary $\mathcal{A}$ adaptively issues queries $\mathcal{O}_{ACH}^{\mathsf{Trapdoor}}$ on a transaction identifier TID. The challenger forwards the trapdoor information $td_{i,\mathsf{TID}}$ associated with user i and the transaction identifier TID to the adversary.

Output $\mathcal{A}$ outputs $(\mathsf{TID}^*, m, r, m', r')$ and succeeds if the following conditions hold.

1. $\mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}^*, m, r) = \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}^*, m', r')$ and $m' \neq m$.
2. $\mathcal{A}$ never queries $\mathcal{O}_{ACH}^{\mathsf{Trapdoor}}$ on TID^*.

Forgery indistinguishability: An ACH scheme is said to be forgery indistinguishable if, for all sufficiently large λ, any $param \leftarrow \mathsf{GlobalSetup}$, $(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(\lambda, param)$, $(pk_j, sk_j) \leftarrow \mathsf{KeyGen}(\lambda, param)$, all transaction identifier TID, and all pairs of messages m and m', the following distribution ensembles are computational indistinguishable:

$$\begin{aligned}\mathcal{D}_{\mathsf{Forge}} &= \{(m', \hat{r}, h) | r \xleftarrow{\$} \mathcal{R}, h \leftarrow \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m, r), td_{i,\mathsf{TID}} \leftarrow \mathsf{Trapdoor}(sk_i, \mathsf{TID}),\\ &\qquad \hat{r} \leftarrow \mathsf{Forge}(sk_j, td_{i,\mathsf{TID}}, pk_i, pk_j, \mathsf{TID}, m, r, h, m')\}_{\lambda, param, pk_i, pk_j, \mathsf{TID}},\\ \mathcal{D}_{\mathsf{Hash}} &= \{(m', r', h') | r' \xleftarrow{\$} \mathcal{R}, h' \leftarrow \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m', r')\}_{\lambda, param, pk_i, pk_j, \mathsf{TID}}.\end{aligned}$$

4.2 Construction

Our specific construction of an ACH scheme consists of the following algorithms:

$\mathsf{GlobalSetup}$ Given a security parameter λ, it first generates a bilinear map group system $\langle p, \mathbb{G}, \mathbb{G}_T, e\rangle$. Then, it picks a generator g of $\mathbb{G}$ and chooses a cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}$. The common public parameter is

$$param = (p, \mathbb{G}, \mathbb{G}_T, e, g, H).$$

KeyGen Given a security parameter λ and the common public parameter $param$, user i first chooses $x_i \in \mathbb{Z}_p^*$ randomly. Then, set his public key as

$$pk_i = g^{x_i},$$

and the private key as $sk_i = x_i$.

Hash Given $param$, user i's public key pk_i, user j's public key pk_j, a message $m \in \mathbb{Z}_p^*$ and a unique transaction identifier TID, it chooses $R \in \mathbb{G}$ uniformly at random and computes

$$h = e(R, g) \cdot e(H(\mathsf{TID})^m, pk_i \cdot pk_j),$$

Finally, it outputs the hash value h.

$\mathsf{Trapdoor}$ Given user i's private key $sk_i = x_i$ and a transaction identifier TID, it computes

$$td_{i,\mathsf{TID}} = H(\mathsf{TID})^{sk_i} = H(\mathsf{TID})^{x_i},$$

and outputs the trapdoor information $td_{i,\mathsf{TID}}$ associated with user i and transaction identifier TID.

Forge Given user j's private key sk_j, the trapdoor information $td_{i,\mathsf{TID}}$ associated with user i and transaction identifier TID, the hash value h on a message m with user i's public key pk_i, user j's public key pk_j, transaction identifier TID, random R, and a message m', it computes and outputs

$$R' = R \cdot (H(\mathsf{TID})^{sk_j} \cdot td_{i,\mathsf{TID}})^{m-m'}.$$

Note that, for a forgery, we have

$$\begin{aligned}\mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m', R') &= e(R', g) \cdot e(H(\mathsf{TID})^{m'}, pk_i \cdot pk_j)\\ &= e(R \cdot (H(\mathsf{TID})^{sk_j} \cdot td_{i,\mathsf{TID}})^{m-m'}, g) \cdot e(H(\mathsf{TID})^{m'}, pk_i \cdot pk_j)\\ &= e(R, g) \cdot e((H(\mathsf{TID})^{sk_j} \cdot td_{i,\mathsf{TID}})^{m-m'}, g) \cdot e(H(\mathsf{TID})^{m'}, pk_i \cdot pk_j)\\ &= e(R, g) \cdot e((H(\mathsf{TID})^{x_j} \cdot H(\mathsf{TID})^{x_i})^{m-m'}, g) \cdot e(H(\mathsf{TID})^{m'}, g^{x_i+x_j})\\ &= e(R, g) \cdot e((H(\mathsf{TID})^{x_i+x_j})^{m-m'}, g) \cdot e((H(\mathsf{TID})^{x_i+x_j})^{m'}, g)\\ &= e(R, g) \cdot e((H(\mathsf{TID})^{x_i+x_j})^{m}, g),\\ &= e(R, g) \cdot e(H(\mathsf{TID})^{m}, g^{x_i+x_j})\\ &= e(R, g) \cdot e(H(\mathsf{TID})^{m}, pk_i \cdot pk_j)\\ &= \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m, R).\end{aligned}$$

So, the above scheme satisfies correctness. We now turn to security.

Theorem 1 *In the random oracle model, the above construction of accountable CH is secure against (existential) collision forgery under active attacks, assuming that the CDH assumption holds in the bilinear map group system* $\langle p, \mathbb{G}, \mathbb{G}_T, e\rangle$.

Proof. Suppose there exists an adversary $\mathcal{A}$ against the collision resistance of above accountable CH scheme. We are going to construct another PPT $\mathcal{B}$ that makes use of $\mathcal{A}$ to solve the CDH problem in a bilinear map group system $\langle p, \mathbb{G}, \mathbb{G}_T, e\rangle$ with non-negligible probability.

$\mathcal{B}$ is given as input a random 3-tuple (g, g^a, g^b), and $\mathcal{B}$'s goal is to compute g^{ab}. $\mathcal{B}$ runs $\mathcal{A}$ executing the following steps.

Setup $\mathcal{B}$ first chooses $x_j \in \mathbb{Z}_p^*$ randomly. Then, set the common public parameters as $param = (p, \mathbb{G}, \mathbb{G}_T, e,\ g, H)$, user i's public key as $pk_i = g^a$, and user j's public/private key pair as $(pk_j, sk_j) = (g^{x_j}, x_j)$. The user i's private key $sk_i = a$, which is unknown to $\mathcal{B}$. Here H is a random oracle controlled by $\mathcal{B}$. $\mathcal{B}$ also maintains a list L which is initially empty. Finally, $\mathcal{B}$ sends $(param, pk_i, (pk_j, sk_j))$ to the adversary $\mathcal{A}$.

Query phase The adversary $\mathcal{A}$ adaptively issues queries:

1. H query on a transaction identifier TID: $\mathcal{B}$ responds as follows:
   - If a tuple $(\mathsf{TID}, Q, t, coin)$ appears in the list L, $\mathcal{B}$ forwards Q to $\mathcal{A}$.
   - Otherwise, $\mathcal{B}$ picks $coin \in \{0, 1\}$ at random such that $\mathsf{Pr}[coin = 0] = \rho$. ($\rho$ will be determined later.) Then $\mathcal{B}$ chooses $t \in \mathbb{Z}_p^*$ randomly and computes $Q = (g^b)^{coin} \cdot g^t$. Finally, the tuple $(\mathsf{TID}, Q, t, coin)$ is added to the list L and Q is sent to $\mathcal{A}$.
2. $\mathcal{O}_{ACH}^{\mathsf{Trapdoor}}$ on a transaction identifier TID: Let $(\mathsf{TID}, Q, t, coin)$ be the corresponding tuple in the list L. $\mathcal{B}$ responds as follows:
   - If $coin = 0$, $\mathcal{B}$ computes $td_{i,\mathsf{TID}} = (g^a)^t$ and sends $td_{i,\mathsf{TID}}$ to $\mathcal{A}$. Observe that $td_{i,\mathsf{TID}} = H(\mathsf{TID})^{sk_i}$, which is the trapdoor information associated with user i and the transaction identifier TID.
   - Otherwise, $\mathcal{B}$ aborts and terminates.

Output $\mathcal{A}$ outputs $(\mathsf{TID}^*, m, R, m', R')$ such that

$$\mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}^*, m, R) = \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}^*, m', R') \text{ and } m' \neq m.$$

Let $(\mathsf{TID}^*, Q^*, t^*, coin^*)$ be the corresponding tuple in the list L. $\mathcal{B}$ proceeds as follows:

- If $coin^* = 0$, $\mathcal{B}$ aborts and terminates.
- Otherwise, $\mathcal{B}$ computes and outputs
$$\frac{(R/R')^{1/(m'-m)}}{(g^b g^{t^*})^{x_j} \cdot (g^a)^{t^*}}.$$

Observe that,

$$\mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}^*, m, R) = e(R, g) \cdot e(H(\mathsf{TID}^*)^m, pk_i \cdot pk_j) = e(R \cdot (H(\mathsf{TID}^*)^{x_i+x_j})^m, g),$$

and

$$\mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}^*, m', R') = e(R', g) \cdot e(H(\mathsf{TID}^*)^{m'}, pk_i \cdot pk_j) = e(R' \cdot (H(\mathsf{TID}^*)^{x_i+x_j})^{m'}, g).$$

If $\mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}^*, m, R) = \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}^*, m', R')$, we have

$$\begin{aligned} R \cdot (H(\mathsf{TID}^*)^{x_i+x_j})^m &= R' \cdot (H(\mathsf{TID}^*)^{x_i+x_j})^{m'}, \\ (R/R')^{1/(m'-m)} &= H(\mathsf{TID}^*)^{x_i+x_j}. \end{aligned}$$

If $coin^* = 1$, $H(\mathsf{TID}^*) = g^b g^{t^*}$; hence

$$\begin{aligned} (R/R')^{1/(m'-m)} &= (g^b g^{t^*})^{a+x_j}, \\ \frac{(R/R')^{1/(m'-m)}}{(g^b g^{t^*})^{x_j} \cdot (g^a)^{t^*}} &= g^{ab}. \end{aligned}$$

Therefore, if $\mathcal{B}$ does not abort during the simulation, $\mathcal{B}$ will be able to solve the CDH problem. Observe that the probability that $\mathcal{B}$ does not abort during the simulation is given by $\rho^q(1-\rho)$ which is maximized at $\rho = 1-1/(q+1)$, where q is the number of $\mathcal{O}_{ACH}^{\mathsf{Trapdoor}}$ queries by the adversary $\mathcal{A}$. Hence, the probability that $\mathcal{B}$ does not abort is at most $1/(\tau(q+1))$, where τ denotes the base of the natural logarithm. This completes the proof.

Theorem 2 *The above construction of ACH is forgery indistinguishable.*

Proof. The ACH scheme is said to be forgery indistinguishable, if for all sufficiently large λ, any $param \leftarrow \mathsf{GlobalSetup}$, $(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(\lambda, param)$, $(pk_j, sk_j) \leftarrow \mathsf{KeyGen}(\lambda, param)$, all transaction identifier TID, and all pairs of messages m and m', the following distribution ensembles are computational indistinguishable:

$$\begin{aligned}\mathcal{D}_{\mathsf{Forge}} &= \{(m', \hat{R}, h) | R \xleftarrow{\$} \mathcal{R}, h \leftarrow \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m, R), td_{i,\mathsf{TID}} \leftarrow \mathsf{Trapdoor}(sk_i, \mathsf{TID}),\\ &\qquad \hat{R} \leftarrow \mathsf{Forge}(sk_j, td_{i,\mathsf{TID}}, pk_i, pk_j, \mathsf{TID}, m, R, h, m')\}_{\lambda, param, pk_i, pk_j, \mathsf{TID}},\\ \mathcal{D}_{\mathsf{Hash}} &= \{(m', R', h') | R' \xleftarrow{\$} \mathcal{R}, h' \leftarrow \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m', R')\}_{\lambda, param, pk_i, pk_j, \mathsf{TID}}.\end{aligned}$$

In our ACH construction, because of the non-degeneracy of bilinear pairing, given a hash value $h = \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m, R)$, there is exactly one random element $\hat{R} \in \mathbb{G}$, such that

$$\mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m, R) = \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}, m', \hat{R}).$$

Since R is chosen uniformly, $\hat{R}$ is also distributed uniformly and the forgery indistinguishability of our construction follows.

5 Generic Construction of ATSS from ACH

Based on IBCH, Canard et al. [8] proposed a generic construction of TSS. In their construction, to sign a message $m = m_1 \| \cdots \| m_L$, the signer first sets $\tilde{m} = \tilde{m}_1 \| \cdots \| \tilde{m}_L$, where $\tilde{m}_i = m_i$ if $i \notin I$ and otherwise, $\tilde{m}_i = h_i = \mathsf{IBCH.Hash}(param, \mathsf{ID}, m_i, r_i)$. The set of indices $I \subseteq [1, L]$ that are sanitizable and the identity ID associated with the transaction are generated by the signer. Then, the signer signs the message $\tilde{m}$ using a conventional signature scheme. Obviously, an entity with the trapdoor associated with ID generated by the signer can modify m_i and generate a new signature on the sanitized message. Our construction of ATSS is similar to the construction proposed by Canard et al. [8], but in order to achieve accountability we use ACH in place of IBCH. In order to generate a new signature on a sanitized message, the sanitizer need to use his private key and a trapdoor information associated with the transaction identifier generated by the signer to find a collision of the ACH. The signer can then use the collision to convince a trusted third party that a message is sanitized, as nobody apart from the sanitizer has more than a negligible probability of successfully finding a second message that produces the same signing value.

Now, given a regular signature scheme $\Sigma = (\Sigma.\mathsf{KeyGen}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Verify})$, and an ACH scheme $\Pi = (\Pi.\mathsf{GlobalSetup}, \Pi.\mathsf{KeyGen}, \Pi.\mathsf{Hash}, \Pi.\mathsf{Trapdoor}, \Pi.\mathsf{Forge})$, we define the 8-tuple algorithms ($\mathsf{GlobalSetup}$, KeyGen, Sign, $\mathsf{Trapdoor}$, $\mathsf{Sanitize}$, Verify, Proof, Judge) of an ATSS scheme as follows:

$\mathsf{GlobalSetup}$ Given a security parameter λ, it first runs

$$param_\Pi \leftarrow \Pi.\mathsf{GlobalSetup}(\lambda),$$

and chooses two cryptographic hash functions $H_1 : \{0,1\}^* \rightarrow \{0,1\}^\lambda, H_2 : \{0,1\}^* \rightarrow \mathcal{R}$. Then, it publishes the common public parameter $param = (param_\Pi, H_1, H_2)$.

KeyGen Given a security parameter λ and the common public parameter $param$, it first runs

$$(pk_\Sigma, sk_\Sigma) \leftarrow \Sigma.\mathsf{KeyGen}(\lambda),\ (pk_\Pi, sk_\Pi) \leftarrow \Pi.\mathsf{KeyGen}(\lambda, param_\Pi).$$

Then, it picks a key $\kappa_{sig} \in \{0,1\}^\lambda$ for the hash function H_1, sets the public key $pk = (pk_\Sigma, pk_\Pi)$ and the private key $sk = (sk_\Sigma, sk_\Pi, \kappa_{sig})$. Finally, it publishes pk and keeps sk secret. We denote by $pk_{sig} = (pk_{sig,\Sigma}, pk_{sig,\Pi})$ and $sk_{sig} = (sk_{sig,\Sigma}, sk_{sig,\Pi}, \kappa_{sig})$ the public key and private key of the signer, and by $pk_{san} = (pk_{san,\Sigma}, pk_{san,\Pi})$ and $sk_{san} = (sk_{san,\Sigma}, sk_{san,\Pi}, \kappa_{san})$ the public key and private key of a sanitizer.

Sign Given a sanitizer's public key $pk_{san} = (pk_{san,\Sigma}, pk_{san,\Pi})$, a message $m = m_1\|\cdots\|m_L$, a set of indices $I \subseteq [1, L]$ that are sanitizable, a transaction identifier TID and the signer's private key $sk_{sig} = (sk_{sig,\Sigma}, sk_{sig,\Pi}, \kappa_{sig})$, it proceeds as follows.

1. Compute $z = H_1(\kappa_{sig}, \mathsf{TID})$.
2. For all $i \in [1, L]\backslash I$, set $\tilde{m}_i = m_i$.
3. For all $i \in I$, compute $r_i = H_2(z, i)$ and
$$h_i = \Pi.\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}, m_i, r_i)$$
and set $\tilde{m}_i = h_i$. Let r be the concatenation of all r_i, $i \in I$.
4. Set $\tilde{m} = \tilde{m}_1\|\cdots\|\tilde{m}_L$ and run
$$\tilde{\sigma} \leftarrow \Sigma.\mathsf{Sign}(\tilde{m}, sk_{sig,\Sigma}).$$
5. Finally, set $\sigma = \tilde{\sigma}\|r$ and output the signature σ on m.

Trapdoor Given a message m, a set of the indices I that are sanitizable, a transaction identifier TID, a valid signature σ on $(pk_{sig}, pk_{san}, m, I, \mathsf{TID})$ and the signer's private key $sk_{sig} = (sk_{sig,\Sigma}, sk_{sig,\Pi}, \kappa_{sig})$, it runs
$$td_{\mathsf{TID}} \leftarrow \Pi.\mathsf{Trapdoor}(sk_{sig,\Pi}, \mathsf{TID}),$$
and outputs the trapdoor td_{TID} associated with TID.

Sanitize Given the signer's public key $pk_{sig} = (pk_{sig,\Sigma}, pk_{sig,\Pi})$, a message $m = m_1\|\cdots\|m_L$, a set of indices $I \subseteq [1, L]$ that are sanitizable, the transaction identifier TID, a valid signature $\sigma = \tilde{\sigma}\|r$ on $(pk_{sig}, pk_{san} = (pk_{san,\Sigma}, pk_{san,\Pi}), m, I, \mathsf{TID})$, a trapdoor td_{TID} associated with TID, the sanitizer's private key $sk_{san} = (sk_{san,\Sigma}, sk_{san,\Pi}, \kappa_{san})$ and a new message $m' = m'_1\|\cdots\|m'_L$, it proceeds as follows.

1. Let $I' = \{i \in [1, L] | m_i \neq m'_i\}$. Check whether $I' \subseteq I$. If not, output $\perp$, denoted an error.
2. Retrieve $\{r_i, i \in I\}$ from the signature $\sigma = \tilde{\sigma}\|r$.
3. For all $i \in I'$, compute $h_i \leftarrow \Pi.\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}, m_i, r_i)$ and
$$r'_i \leftarrow \Pi.\mathsf{Forge}(sk_{san,\Pi}, td_{\mathsf{TID}}, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}, m_i, r_i, h_i, m'_i).$$
4. For all $i \in I\backslash I'$, set $r'_i = r_i$. Let r' be the concatenation of all r'_i, $i \in I$.
5. Set $\sigma' = \tilde{\sigma}\|r'$ and output the new signature σ' on $(pk_{sig}, pk_{san}, m', I, \mathsf{TID})$.

Verify Given $param$, the signer's public key $pk_{sig} = (pk_{sig,\Sigma}, pk_{sig,\Pi})$, a sanitizer's public key $pk_{san} = (pk_{san,\Sigma}, pk_{san,\Pi})$, a message $m = m_1\|\cdots\|m_L$, a set of indices $I \subseteq [1, L]$ that are sanitizable, a transaction identifier TID and a putative signature $\sigma = \tilde{\sigma}\|r$, it proceeds as follows.

1. Retrieve $\{r_i, i \in I\}$ from the signature $\sigma = \tilde{\sigma}\|r$.
2. For all $i \in [1, L]\backslash I$, set $\tilde{m}_i = m_i$.
3. For all $i \in I$, compute
$$h_i = \Pi.\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}, m_i, r_i)$$
and set $\tilde{m}_i = h_i$.
4. Set $\tilde{m} = \tilde{m}_1\|\cdots\|\tilde{m}_L$ and output $\Sigma.\mathsf{Verify}(pk_{sig,\Sigma}, \tilde{m}, \tilde{\sigma})$.

Proof Given $param$, the signer's private key $sk_{sig} = (sk_{sig,\Sigma}, sk_{sig,\Pi}, \kappa_{sig})$, a valid message-signature pair $((pk_{sig} = (pk_{sig,\Sigma}, pk_{sig,\Pi}), pk_{san} = (pk_{san,\Sigma}, pk_{san,\Pi}), m, I, \mathsf{TID}), \sigma)$, and a set of (polynomially many) additional message-signature pairs $MesSigS = ((pk_{sig}, pk^{(i)}_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})_{i=1,2,\ldots,q}$ generated originally by the signer, it first searches the set $MesSigS$ to find a tuple $((pk_{sig}, pk^{(i)}_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})$ such that

1. $pk_{san} = pk^{(i)}_{san}$, $I = I^{(i)}$ and $\mathsf{TID} = \mathsf{TID}^{(i)}$.

2. $I' \subseteq I$, where $I' = \{j \in [1, L] | m_j \neq m_j^{(i)}\}$. Note that, $m = m_1 \| \cdots \| m_L$ and $m^{(i)} = m_1^{(i)} \| \cdots \| m_L^{(i)}$.
3. $\Pi.\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}, m_j, r_j) = \Pi.\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}^{(i)}, \mathsf{TID}, m_j^{(i)}, r_j^{(i)})$ for all $j \in I'$, where $\sigma = \tilde{\sigma} \| r$, $r = \{r_j, j \in I\}$, $\sigma^{(i)} = \tilde{\sigma} \| r^{(i)}$ and $r^{(i)} = \{r_j^{(i)}, j \in I\}$.

Then, it computes $z_i = H_1(\mathsf{TID}^{(i)}, \kappa_{sig})$. Finally, it outputs the proof $\pi = (pk_{sig}, pk_{san}^{(i)}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}, \sigma^{(i)}, z_i)$.

Judge Given $param$, the signer's public key $pk_{sig} = (pk_{sig,\Sigma}, pk_{sig,\Pi})$, the sanitizer's public key $pk_{san} = (pk_{san,\Sigma}, pk_{san,\Pi})$, a valid message-signature pair $((pk_{sig}, pk_{san}, m, I, \mathsf{TID}), \sigma)$ and a proof $\pi = (pk_{sig}, pk_{san}^{(i)}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}, \sigma^{(i)}, z_i)$. Let $\sigma = \tilde{\sigma} \| r$ where $r = \{r_j, j \in I\}$, and $\sigma^{(i)} = \tilde{\sigma} \| r^{(i)}$ where $r^{(i)} = \{r_j^{(i)}, j \in I\}$, it first checks whether the following conditions hold:

1. $pk_{san} = pk_{san}^{(i)}$, $I = I^{(i)}$ and $\mathsf{TID} = \mathsf{TID}^{(i)}$;
2. $(m, \sigma) \neq (m^{(i)}, \sigma^{(i)})$;
3. $I' \subseteq I$, where $I' = \{j \in [1, L] | m_j \neq m_j^{(i)}\}$. Note that, $m = m_1 \| \cdots \| m_L$ and $m^{(i)} = m_1^{(i)} \| \cdots \| m_L^{(i)}$;
4. $\Pi.\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}, m_j, r_j) = \Pi.\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}, m_j^{(i)}, r_j^{(i)})$ for all $j \in I'$;
5. $r_j^{(i)} = H_2(z_i, j)$ for all $j \in I$.

If so, it outputs `San` indicating the message-signature pair $((pk_{sig}, pk_{san}, m, I, \mathsf{TID}), \sigma)$ was created by the sanitizer; else it outputs `Sig` indicating the message-signature pair $((pk_{sig}, pk_{san}, m, I, \mathsf{TID}), \sigma)$ was created by the signer.

It is obvious that the above ATSS scheme satisfies correctness. We now state the security theorems of the scheme, including *unforgeability*, *indistinguishability* and *accountability*.

Theorem 3 (Unforgeability) *If the signature scheme Σ is existential unforgeable under adaptive chosen message attacks and the ACH scheme Π is resistant to collision forgery under active attacks, the above construction of ATSS is existential unforgeable under adaptive chosen message attacks.*

Proof. As usual [8], for presentation simplicity, we assume that a message consists of a single block (extension to the general case is straightforward). Let $\mathcal{A}$ be a forger against the above ATSS scheme and let $(m^*, I^* = \{1\}, \mathsf{TID}^*, \sigma^*)$ be the forgery produced by $\mathcal{A}$. We distinguish between two cases of forgery:

Case 1: m^* is a non-sanitized message and $\mathcal{A}$ never queries $\mathcal{O}_{ATSS}^{\mathsf{Sign}}$ oracle on $(m^*, I^*, \mathsf{TID}^*)$.

Case 2: m^* is a sanitized message. A never queries $\mathcal{O}_{ATSS}^{\mathsf{Sanitize}}$ on $(m, I^*, \mathsf{TID}^*, \sigma, m^*)$ or $\mathcal{O}_{ATSS}^{\mathsf{Trapdoor}}$ on $(m, I^*, \mathsf{TID}^*, \sigma)$, where σ is a valid signature on $(pk_{sig}, pk_{san}, m, I^*, \mathsf{TID}^*)$ and $m \neq m^*$.

To prove this theorem, we will show that in the first case, we can use $\mathcal{A}$ as a sub-routine to construct another algorithm $\mathcal{B}$, which is a forgery against the signature scheme Σ, and in the second case, we can use $\mathcal{A}$ as a sub-routine to construct another algorithm $\mathcal{B}$, which is a forgery against the ACH scheme Π. Initially, $\mathcal{B}$ will choose a random bit $b \in \{1, 2\}$ that indicates its guess for the case of forgery that $\mathcal{A}$ will produce. $\mathcal{B}$ proceeds differently for each case.

Case 1. Given a public key pk^* and a signing oracle $\mathcal{O}_{\Sigma}^{\mathsf{Sign}}$ of the signature scheme Σ, using $\mathcal{A}$ as a sub-routine, $\mathcal{B}$ simulates a forger against the signature scheme Σ. $\mathcal{B}$ proceeds as follows.

Setup $\mathcal{B}$ first runs $\Pi.\mathsf{Setup}$ algorithm to generate a pair of keys (pk_Π, sk_Π) and picks a key $\kappa_{sig} \in \{0, 1\}^\lambda$ for the hash function H_1. Then, $\mathcal{B}$ sets the signer's public key pk_{sig} of the above ATSS scheme as $pk_{sig} = (pk^*, pk_\Pi)$. The signer's private key $sk_{sig} = (sk^*, sk_\Pi, \kappa_{sig})$ and sk^* is unknown to $\mathcal{B}$. Next, $\mathcal{B}$ runs KeyGen algorithm of the above ATSS to generate a pair of key

$$(pk_{san} = (pk_{san,\Sigma}, pk_{san,\Pi}),\ sk_{san} = (sk_{san,\Sigma}, sk_{san,\Pi}, \kappa_{san}))$$

for the sanitizer. Finally, $\mathcal{B}$ gives the signer's public key pk_{sig} and the sanitizer's public key pk_{san} to $\mathcal{A}$.

Query phase $\mathcal{A}$ adaptively issues queries:

1. $\mathcal{O}_{ATSS}^{\mathsf{Sign}}$ queries: $\mathcal{B}$ runs Sign algorithm of the above ATSS scheme with the knowledge of κ_{sig} and the help of its $\mathcal{O}_{\Sigma}^{\mathsf{Sign}}$ oracle, and sends the results to $\mathcal{A}$.
2. $\mathcal{O}_{ATSS}^{\mathsf{Trapdoor}}$ queries: $\mathcal{B}$ runs $\mathsf{Trapdoor}$ algorithm of the above ATSS scheme with the knowledge of sk_{Π}, and sends the results to $\mathcal{A}$.
3. $\mathcal{O}_{ATSS}^{\mathsf{Sanitize}}$ queries: $\mathcal{B}$ runs $\mathsf{Sanitize}$ algorithm of the above ATSS scheme with the knowledge of sk_{Π} and sk_{san}, and sends the results to $\mathcal{A}$.

Output $\mathcal{A}$ outputs a forgery $(m^*, I^* = \{1\}, \mathsf{TID}^*, \sigma^* = \tilde{\sigma}^* \| r^*)$ against the above ATSS scheme. $\mathcal{B}$ proceeds as follows.

1. Computes $h^* = \Pi.\mathsf{Hash}(param, pk_{\Pi}, pk_{san,\Pi}, \mathsf{TID}^*, m^*, r^*)$ and set $\tilde{m}^* = h^*$.
2. Output $(\tilde{m}^*, \tilde{\sigma}^*)$.

It is obvious that if $(m^*, I^*, \mathsf{TID}^*, \sigma^* = \tilde{\sigma}^* \| r^*)$ is a successful forgery against the above ATSS scheme, $(\tilde{m}^*, \tilde{\sigma}^*)$ is a successful forgery against the signature scheme Σ.

Case 2. Given user i's public key pk_i, user j's public/private key pair (pk_j, sk_j) and a trapdoor generation oracle $\mathcal{O}_{\Pi}^{\mathsf{Trapdoor}}$ of the ACH scheme Π, using $\mathcal{A}$ as a sub-routine, $\mathcal{B}$ simulates a forger against the ACH scheme Π. $\mathcal{B}$ proceeds as follows.

Setup $\mathcal{B}$ first runs $\Sigma.\mathsf{KeyGen}$ algorithm to generate a pair of keys $(pk_{\Sigma}, sk_{\Sigma})$ and picks a key $\kappa_{sig} \in \{0,1\}^{\lambda}$ for the hash function H_1. Then, $\mathcal{B}$ sets the signer's public key pk_{sig} of the above ATSS scheme as $pk_{sig} = (pk_{\Sigma}, pk_i)$. The signer's private key $sk_{sig} = (sk_{\Sigma}, sk_i, \kappa_{sig})$ and sk_i are unknown to $\mathcal{B}$. Next, $\mathcal{B}$ runs $\Sigma.\mathsf{KeyGen}$ algorithm to obtain another pair of key $(pk'_{\Sigma}, sk'_{\Sigma})$ and picks another key $\kappa_{san} \in \{0,1\}^{\lambda}$ for the hash function H_1, and sets the sanitizer's public/private key pair as

$$pk_{san} = (pk'_{\Sigma}, pk_j), \ sk_{san} = (sk'_{\Sigma}, sk_j, \kappa_{san}).$$

Finally, $\mathcal{B}$ gives the signer's public key pk_{sig} and the sanitizer's public key pk_{san} to $\mathcal{A}$.

Query phase $\mathcal{A}$ adaptively issues the following queries:

1. $\mathcal{O}_{ATSS}^{\mathsf{Sign}}$ queries: $\mathcal{B}$ runs Sign algorithm of the above ATSS scheme with the knowledge of sk_{Σ} and κ_{sig}, and sends the results to $\mathcal{A}$.
2. $\mathcal{O}_{ATSS}^{\mathsf{Trapdoor}}$ queries: $\mathcal{B}$ runs $\mathsf{Trapdoor}$ algorithm of the above ATSS scheme with the help of its $\mathcal{O}_{\Pi}^{\mathsf{Trapdoor}}$ oracle, and sends the results to $\mathcal{A}$.
3. $\mathcal{O}_{ATSS}^{\mathsf{Sanitize}}$ queries: $\mathcal{B}$ runs $\mathsf{Sanitize}$ algorithm of the above ATSS scheme with the help of its $\mathcal{O}_{\Pi}^{\mathsf{Trapdoor}}$ oracle and the knowledge of sk_{san}, and sends the results to $\mathcal{A}$.

Output $\mathcal{A}$ outputs a forgery $(m^*, I^*, \mathsf{TID}^*, \sigma^* = \tilde{\sigma}^* \| r^*)$ against the above ATSS scheme. Note that, in this case, m^* is a sanitized message. So, there exists a message m, corresponding to a signature $\sigma = \tilde{\sigma}^* \| r$, which has been asked to $\mathcal{O}_{ATSS}^{\mathsf{Sign}}$ oracle, such that

$$\mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}^*, m^*, r^*) = \mathsf{Hash}(param, pk_i, pk_j, \mathsf{TID}^*, m, r).$$

$\mathcal{B}$ outputs $(\mathsf{TID}^*, m^*, r^*, m, r)$.

It is obvious that if $(m^*, I^*, \mathsf{TID}^*, \sigma^* = \tilde{\sigma}^* \| r^*)$ is a successful forgery against the above ATSS scheme, $(\mathsf{TID}^*, m^*, r^*, m, r)$ is a collision against the ACH scheme Π.

This completes the proof.

Theorem 4 (Indistinguishability) *If the ACH scheme Π is forgery indistinguishable, in the random oracle model, the following distributions $\mathcal{D}_{\mathsf{Sanitize}}$ and $\mathcal{D}_{\mathsf{Sign}}$ are computational indistinguishable for all sufficiently large λ, any $param \leftarrow \mathsf{GlobalSetup}$, $(pk_{sig}, sk_{sig}) \leftarrow \mathsf{KeyGen}(\lambda, param)$, $(pk_{san}, sk_{san}) \leftarrow \mathsf{KeyGen}(\lambda, param)$, any set of indices $I \subseteq [1, L]$, any message pairs m, m' such that $m_i = m'_i$ for all $i \notin I$, and any transaction identifier TID:*

$$\begin{aligned}
\mathcal{D}_{\mathsf{Sanitize}} &= \{(m', \hat{\sigma}) | \sigma \leftarrow \mathsf{Sign}(pk_{san}, m, I, \mathsf{TID}, sk_{sig}), td_{\mathsf{TID}} \leftarrow \mathsf{Trapdoor}(m, I, \mathsf{TID}, \sigma, sk_{sig}), \\
&\qquad \hat{\sigma} \leftarrow \mathsf{Sanitize}(pk_{sig}, m, I, \mathsf{TID}, \sigma, m', td_{\mathsf{TID}}, sk_{san})\}_{\lambda, param, pk_{sig}, pk_{san}, I, \mathsf{TID}}, \\
\mathcal{D}_{\mathsf{Sign}} &= \{(m', \sigma') | \sigma' \leftarrow \mathsf{Sign}(pk_{san}, m', I, \mathsf{TID}, sk_{sig})\}_{\lambda, param, pk_{sig}, pk_{san}, I, \mathsf{TID}}.
\end{aligned}$$

Proof. To keep the presentation compact and without loss of generality, we assume that messages m and m' consists of a single block. In the above ATSS scheme, $\mathcal{D}_{\mathsf{Sanitize}} = \{(m', \hat{\sigma} = \tilde{\sigma} \| \hat{r})\}_{\lambda, param, pk_{sig}, pk_{san}, I, \mathsf{TID}}$ and $\mathcal{D}_{\mathsf{Sign}} = \{(m', \sigma' = \tilde{\sigma}' \| r')\}_{\lambda, param, pk_{sig}, pk_{san}, I, \mathsf{TID}}$, where

$$r \xleftarrow{\$} \mathcal{R},\ h \leftarrow \Pi.\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}, m, r),\ \tilde{\sigma} \leftarrow \Sigma.\mathsf{Sign}(h, sk_{sig,\Sigma}),$$
$$\Pi.\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}, m, r) = \Pi.\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}, m', \hat{r}),$$

and

$$r' \xleftarrow{\$} \mathcal{R},\ h' \leftarrow \Pi.\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}, m, r),\ \tilde{\sigma}' \leftarrow \Sigma.\mathsf{Sign}(h', sk_{sig,\Sigma}).$$

Note that, $r \xleftarrow{\$} \mathcal{R}$ and $r' \xleftarrow{\$} \mathcal{R}$ are derived from the assumption that H_2 is a random oracle. Because Π is forgery indistinguishable, it is obvious that $\mathcal{D}_{\mathsf{Sanitize}}$ and $\mathcal{D}_{\mathsf{Sign}}$ are indistinguishable. This completes the proof.

Theorem 5 (Sanitizer-accountability) *If the signature scheme Σ is existential unforgeable under adaptive chosen message attacks, the above construction of ATSS is sanitizer-accountable.*

Proof. Assume that the scheme is not sanitizer-accountable and there is a successful adversary $\mathcal{A}$. We will show that we can use $\mathcal{A}$ as a sub-routine to construct another algorithm $\mathcal{B}$, which is a forgery against the signature scheme Σ. Again, without loss of generality, we assume that a message consists of a single block.

Given a public key pk^* and a signing oracle $\mathcal{O}_{\Sigma}^{\mathsf{Sign}}$ of the signature scheme Σ, using $\mathcal{A}$ as a sub-routine, $\mathcal{B}$ simulates a forger against the signature scheme Σ. $\mathcal{B}$ proceeds as follows.

Setup $\mathcal{B}$ first runs $\Pi.\mathsf{Setup}$ algorithm to generate a pair of keys (pk_{Π}, sk_{Π}) and picks a key $\kappa_{sig} \in \{0,1\}^{\lambda}$ for the hash function H_1. Then, $\mathcal{B}$ sets the signer's public key pk_{sig} of the above ATSS scheme as $pk_{sig} = (pk^*, pk_{\Pi})$. The signer's private key $sk_{sig} = (sk^*, sk_{\Pi}, \kappa_{sig})$ and sk^* is unknown to $\mathcal{B}$. Finally, $\mathcal{B}$ gives the signer's public key pk_{sig} to $\mathcal{A}$.

Query phase $\mathcal{A}$ adaptively issues queries:

1. $\mathcal{O}_{TSS}^{\mathsf{Sign}}$ queries on $(pk_{san}, m, I, \mathsf{TID})$: $\mathcal{B}$ runs Sign algorithm of the above ATSS scheme with the knowledge of κ_{sig} and the help of its $\mathcal{O}_{\Sigma}^{\mathsf{Sign}}$ oracle, and sends the results to $\mathcal{A}$.
2. $\mathcal{O}_{TSS}^{\mathsf{Trapdoor}}$ queries on $(pk_{san}, m, I, \mathsf{TID}, \sigma)$: $\mathcal{B}$ runs $\mathsf{Trapdoor}$ algorithm of the above ATSS scheme with the knowledge of sk_{Π}, and sends the results to $\mathcal{A}$.

Output $\mathcal{A}$ outputs $(pk_{sig}, pk^*_{san} = (pk^*_{san,\Sigma}, pk^*_{san,\Pi}), m^*, I^* = \{1\}, \mathsf{TID}^*, \sigma^* = \tilde{\sigma}^* \| r^*)$ against the sanitizer-accountability of above ATSS scheme. $\mathcal{B}$ does as follows.

1. Compute $h^* = \Pi.\mathsf{Hash}(param, pk_{\Pi}, pk_{san,\Pi}, \mathsf{TID}^*, m^*, r^*)$ and set $\tilde{m}^* = h^*$.
2. Output $(\tilde{m}^*, \tilde{\sigma}^*)$.

Let $(pk^{(i)}_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)})$ and $\sigma^{(i)}$ for $i = 1, 2, \ldots, q$ denote the queries and answers to and from oracle $\mathcal{O}_{ATSS}^{\mathsf{Sign}}$. If $\mathcal{A}$ is a successful adversary, then the message-signature pair $((pk_{sig}, pk^*_{san}, m^*, I^*, \mathsf{TID}^*), \sigma^*)$ is not sanitized from $((pk_{sig}, pk^{(i)}_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})_{i=1,2,\ldots,q}$. Hence, $(\tilde{m}^*, \tilde{\sigma}^*)$ is a successful forgery against the signature scheme Σ. This completes the proof.

Theorem 6 (Signer-accountability) *If the ACH scheme Π is resistant to collision forgery under active attacks, in the random oracle model, the above construction of ATSS is signer-accountable.*

Proof. Assume that the scheme is not signer-accountable and there is a successful adversary $\mathcal{A}$. We will show that we can use $\mathcal{A}$ as a sub-routine to construct another algorithm $\mathcal{B}$, which is a forgery against the ACH scheme Π. As before, we assume that a message consists of a single block. $\mathcal{B}$ proceeds as follows.

Setup $\mathcal{B}$ first runs KeyGen algorithm of the above ATSS to generate a pair of keys

$$(pk_{san} = (pk_{san,\Sigma}, pk_{san,\Pi}),\ sk_{san} = (sk_{san,\Sigma}, sk_{san,\Pi}, \kappa_{san}))$$

for the sanitizer. Then, $\mathcal{B}$ gives the sanitizer's public key pk_{san} to $\mathcal{A}$.

Query phase $\mathcal{A}$ adaptively issues queries $\mathcal{O}_{ATSS}^{\mathsf{Sanitize}}$ query $(pk_{sig}, m, I, \mathsf{TID}, \sigma, td_{\mathsf{TID}}, m')$: $\mathcal{B}$ runs $\mathsf{Sanitize}$ algorithm of the above ATSS scheme with the knowledge of the private key sk_{san} and the trapdoor td_{TID}, and sends the results to $\mathcal{A}$.

Output $\mathcal{A}$ outputs $(pk_{sig}^*, pk_{san}, m^*, I^* = \{1\}, \mathsf{TID}^*, \sigma^*, \pi^*)$ against the signer-accountability of above ATSS scheme. Let $\sigma^* = \tilde{\sigma}^* \| r^*$ and $\pi^* = (pk_{sig}^*, pk_{san}, m', I^* = \{1\}, \mathsf{TID}^*, \sigma' = \tilde{\sigma}' \| r', z')$, $\mathcal{B}$ outputs $(\mathsf{TID}^*, m^*, r^*, m', r')$.

Now, we show that $(\mathsf{TID}^*, m^*, r^*, m', r')$ is a collision against the accountable CH scheme Π. Let $((pk_{sig}^{(i)}, pk_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})$ for $i = 1, 2, \ldots, q$ denote the answers from oracle $\mathcal{O}_{ATSS}^{\mathsf{Sanitize}}$. If $\mathcal{A}$ is a successful adversary, then

1. $((pk_{sig}^*, pk_{san}, m^*, I^*, \mathsf{TID}^*), \sigma^*) \neq ((pk_{sig}^{(i)}, pk_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})_{i=1,2,\ldots,q}$;
2. $\tilde{\sigma}^* = \tilde{\sigma}'$ and $H_2(z', 1) = r'$;
3. $\mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}^*, m^*, r^*) = \mathsf{Hash}(param, pk_{sig,\Pi}, pk_{san,\Pi}, \mathsf{TID}^*, m', r')$.

Note that, $\mathcal{B}$ does not know the trapdoor td_{TID^*} associated with TID^* since $((pk_{sig}^*, pk_{san}, m^*, I^*, \mathsf{TID}^*), \sigma^*) \neq ((pk_{sig}^{(i)}, pk_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})_{i=1,2,\ldots,q}$. Assuming that H_2 is a random oracle, then

$$((pk_{sig}^*, pk_{san}, m', I^*, \mathsf{TID}^*), \sigma') \neq ((pk_{sig}^{(i)}, pk_{san}, m^{(i)}, I^{(i)}, \mathsf{TID}^{(i)}), \sigma^{(i)})_{i=1,2,\ldots,q},$$

with overwhelming probability, because $\mathcal{A}$ cannot find z' such that $H_2(z', 1) = r'$. Therefore, $(\mathsf{TID}^*, m^*, r^*, m', r')$ is a collision against the accountable CH scheme Π. This completes the proof.

6 Conclusion and Future Work

In this paper, we motivated and introduced the notion of accountable trapdoor sanitizable signature (ATSS). As a building block of ATSS and that might be of independent interest, we also introduced the notion of accountable chameleon hash (ACH), which is an extension of chameleon hash. We defined the security requirements for ACH, and proposed a concrete construction that satisfies the requirements based on the CDH assumption in the random oracle model. Finally, Based on ACH, we proposed a generic construction of ATSS. Instantiating the generic construction with our ACH scheme, we constructed the first ATSS scheme. An important future research problem is to construct ACH schemes (and thus accordingly, ATSS schemes) in the standard model.

References

1. Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In *ESORICS*, pages 159–177, 2005.
2. Giuseppe Ateniese and Breno de Medeiros. Identity-based chameleon hash and applications. In *Financial Cryptography*, pages 164–180, 2004.
3. Giuseppe Ateniese and Breno de Medeiros. On the key exposure problem in chameleon hashes. In *SCN*, pages 165–179, 2004.
4. Feng Bao, Robert H. Deng, Xuhua Ding, Junzuo Lai, and Yunlei Zhao. Hierarchical identity-based chameleon hash and its applications. In *ACNS*, pages 201–219, 2011.
5. Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *CRYPTO*, pages 216–233, 1994.
6. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
7. Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In *Public Key Cryptography*, pages 317–336, 2009.
8. Sébastien Canard, Fabien Laguillaumie, and Michel Milhau. Trapdoor sanitizable signatures and their application to content protection. In *ACNS*, pages 258–276, 2008.

9. Xiaofeng Chen, Fangguo Zhang, and Kwangjo Kim. Chameleon hashing without key exposure. In *ISC*, pages 87–98, 2004.
10. Xiaofeng Chen, Fangguo Zhang, Willy Susilo, Haibo Tian, Jin Li, and Kwangjo Kim. Identity-based chameleon hash scheme without key exposure. In *ACISP*, pages 200–215, 2010.
11. Xiaofeng Chen, Fangguo Zhang, Haibo Tian, Baodian Wei, and Kwangjo Kim. Key-exposure free chameleon hashing and signatures based on discrete logarithm systems. Cryptology ePrint Archive, Report 2009/035, 2009. `http://eprint.iacr.org/`.
12. Robert H. Deng and Yanjiang Yang. A study of data authentication in proxy-enabled multimedia delivery systems: Model, schemes and application. *ACM T. on Multimedia Computing, Communications and Applications*, 5(4):28.1–28.20, 2009.
13. Wei Gao, Fei Li, and Xueli Wang. Chameleon hash without key exposure based on schnorr signature. *Computer Standards & Interfaces*, 31(2):282–285, 2009.
14. Wei Gao, Xueli Wang, and Dongqing Xie. Chameleon hashes without key exposure based on factoring. *J. Comput. Sci. Technol.*, 22(1):109–113, 2007.
15. Tetsuya Izu, Nobuyuki Kanaya, Masahiko Takenaka, and Takashi Yoshioka. Piats: A partially sanitizable signature scheme. In *ICICS*, pages 72–83, 2005.
16. Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262, 2002.
17. Marek Klonowski and Anna Lauks. Extended sanitizable signatures. In *ICISC*, pages 343–355, 2006.
18. Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *NDSS*, 2000.
19. Silvio Micali and Ronald L. Rivest. Transitive signature schemes. In *CT-RSA*, pages 236–243, 2002.
20. Kunihiko Miyazaki, Goichiro Hanaoka, and Hideki Imai. Invisibly sanitizable digital signature scheme. *IEICE Transactions*, 91-A(1):392–402, 2008.
21. Kunihiko Miyazaki, Mitsuru Iwamura, Tsutomu Matsumoto, Ryôichi Sasaki, Hiroshi Yoshiura, Satoru Tezuka, and Hideki Imai. Digitally signed document sanitizing scheme with disclosure condition control. *IEICE Transactions*, 88-A(1):239–246, 2005.
22. Henrich Christopher Pöhls, Kai Samelin, and Joachim Posegga. Sanitizable signatures in xml signature - performance, mixing properties, and revisiting the property of transparency. In *ACNS*, pages 166–182, 2011.
23. Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.
24. Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content extraction signatures. In *ICISC*, pages 285–304, 2001.
25. Dae Hyun Yum, Jae Woo Seo, and Pil Joong Lee. Trapdoor sanitizable signatures made easy. In *ACNS*, pages 53–68, 2010.
26. Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. Id-based chameleon hashes from bilinear pairings. Cryptology ePrint Archive, Report 2003/208, 2003. `http://eprint.iacr.org/`.