This is the full version of a paper which appears in Information Security and Privacy -
ACISP 2017 - 22nd Australasian Conference on Information Security and Privacy, Auckland, New Zealand, July 3-5, 2017,
Proceedings. LNCS 10342. pages ???–???. © Springer, 2017.
pp 437-452

Practical Strongly Invisible and Strongly Accountable Sanitizable Signatures

Michael Till Beck[1], Jan Camenisch[2,‡], David Derler[3,‖], Stephan Krenn[4,‖], Henrich C. Pöhls[5,‖], Kai Samelin[2,6,‡], and Daniel Slamanig[3,‖]

[1] Ludwig-Maximilians-Universität München, Munich, Germany
michael.beck@ifi.lmu.de
[2] IBM Research – Zurich, Rüschlikon, Switzerland
{jca|ksa}@zurich.ibm.com
[3] IAIK, Graz University of Technology, Graz, Austria
{david.derler|daniel.slamanig}@tugraz.at
[4] AIT Austrian Institute of Technology GmbH, Vienna, Austria
stephan.krenn@ait.ac.at
[5] ISL & Chair of IT-Security, University of Passau, Passau, Germany
hp@sec.uni-passau.de
[6] TU Darmstadt, Darmstadt, Germany

Abstract. Sanitizable signatures are a variant of digital signatures where a designated party (the sanitizer) can update admissible parts of a signed message. At PKC'17, Camenisch et al. introduced the notion of *invisible* sanitizable signatures that hides from an outsider which parts of a message are admissible. Their security definition of invisibility, however, does not consider dishonest signers. Along the same lines, their signer-accountability definition does not prevent the signer from falsely accusing the sanitizer of having issued a signature on a sanitized message by exploiting the malleability of the signature itself. Both issues may limit the usefulness of their scheme in certain applications.

We revise their definitional framework, and present a new construction eliminating these shortcomings. In contrast to Camenisch et al.'s construction, ours requires only standard building blocks instead of chameleon hashes with ephemeral trapdoors. This makes this, now even stronger, primitive more attractive for practical use. We underpin the practical efficiency of our scheme by concrete benchmarks of a prototype implementation.

1 Introduction

Digital signatures are an important means to protect the integrity and authenticity of digital data. Ordinary digital signatures are all-or-nothing in the sense that once a message has been signed, it is only possible to verify whether the signature is valid for the entire original message or not. In particular, it is not possible to update (parts of) a signed message in a determined manner without invalidating the signature. However, there are many real-life use cases in which a subsequent modification of the signed data by some designated entity is desired. As an illustrative example consider a patient record that is signed by a medical doctor. The accountant, tasked to charge an insurance company, requires authentic information about the treatments and the patient's insurance number, but not of other parts of the patient record. Clearly, when using conventional digital signatures to guarantee the authenticity of the patient record, far too much information is revealed to the accountant. One solution to avoid such privacy intrusive practices would require the doctor to re-sign only the data relevant for the accountant. However, this would have to be repeated every time a new subset of the record needs to be forwarded to some party that demands authentic information. Especially, this would induce too much overhead to be practical in real scenarios, or may even be impossible due to loss of availability of the doctor for signing subsets from old documents.

‡ Supported by EU ERC PERCY , grant agreement n°32131.
‖ Supported by EU H2020 project PRISMACLOUD: This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n°644962.

Sanitizable signature schemes (SSS) [3] allow for such controlled modifications of signed messages without invalidating the signature. In fact, they are more general than strictly needed by the given example: when signing, the signer determines which blocks $m[i]$ of the message $m = (m[1], m[2], \ldots, m[i], \ldots, m[\ell])$ can be updated (i.e., are flagged *admissible*). Any such admissible block can later be changed to a different bitstring $m[i]' \in \{0,1\}^*$, where $i \in \{1, 2, \ldots, \ell\}$, by a designated party named the *sanitizer*. The sanitizer is represented by a public key. The sanitization process requires the corresponding private key, but does not require the signer's involvement. Sanitization of a message m results in an altered message $m' = (m[1]', m[2]', \ldots, m[i]', \ldots, m[\ell]')$, where $m[i] = m[i]'$ for every non-admissible block, and also a signature σ', which verifies under the given public keys. Hence, authenticity of the message m' is still ensured. Coming back to the above example, playing the role of the sanitizer, a server storing the signed patient records is able to black-out the sensitive parts of a signed patient record, without any additional communication with the doctor, and, in particular, without access to doctor's signing key.

Concrete real-world applications of SSSs include secure routing, privacy-preserving document disclosure, anonymous credentials, group content protection, and blank signatures [3, 12, 13, 14, 20, 21, 24, 35].

Motivation. Recently, the property of invisibility was proposed by Camenisch et al. [15] as a very strong notion of privacy for SSS.[7] Informally, this property guarantees that an outsider cannot even decide which blocks of a signed message are admissible. This property is especially useful, if it must be hidden which parts of a signed message can be changed by a sanitizer. However, their invisibility definition is weak in the sense that it is not possible to query the sanitization oracle for keys different from the challenged ones. Thus, as soon as the adversary gains access to a sanitization oracle, it may be able to decide this question, which may be too limiting, or surprising, in certain use-cases. This is in particular relevant, if a sanitizer needs to sanitize messages from multiple signers: in one of their application scenarios, a cloud-service is used to outsource some computations. The results, however, need to be signed by the outsourcing party. Using SSSs, the cloud can sanitize a signed message, and input the result of the computation. However, it must remain hidden which computations are outsourced to protect trade secrets. This is precisely captured by the invisibility property. However, if the cloud-service uses the same key pair for multiple clients, Camenisch et al.'s [15] definition is not sufficient. Moreover, their construction does not achieve "strong signer-accountability", as defined by Krenn et al. [40]. Namely, they do not prevent the signer from exploiting the malleability of previously seen sanitized signatures to accuse the sanitizer of having created one particular signature. We stress that this only addresses the signature; the message in question still needs to be issued by the sanitizer at some point. We stress that this limitation is explicitly mentioned by Camenisch et al. [15]. Nonetheless, lacking both properties may lead to some practical issues. For "strong signer-accountability", this was already pointed out by Krenn et al. [40], and thus it is desirable to achieve the strengthened properties.

Contribution. Our contribution is manifold. (1) We present a strengthened invisibility definition dubbed *strong* invisibility, which even protects against dishonest signers. (2) We present a provably secure construction of strongly invisible sanitizable signatures, which (3) also achieves a stronger accountability notion compared to the construction by Camenisch et al. [15]. In particular, we exclude the malleability of the signatures, even for signers. This makes our construction suitable for a broader range of applications. Moreover, (4) our construction does not require chameleon-hashes with ephemeral trapdoors, and can thus be considered simpler than the one in [15], as only standard primitives are required. In more detail, the construction is solely based on unique signatures, labeled CCA2-secure encryption schemes, and collision-resistant chameleon-hashes, paired with a special and novel way to generate randomness for the chameleon-hashes. Finally, (5) to demonstrate that our construction is practical, we have implemented it. The evaluation shows that the primitive is efficient enough for most use-cases.

[7] Their idea dates back to the original paper by Ateniese et al. [3], which name this property "strong transparency" (cf. Pöhls et al. for a discussion [49]). However, they neither provide a formal definition nor a provably secure construction.

Related Work and State-of-the-Art. SSSs have been introduced by Ateniese et al. [3], and later most of the current security properties were introduced by Brzuska et al. [10] (with some later refinements due to Gong et al. [34]). Later on, additional properties such as (strong) unlinkability [12, 14, 30], and non-interactive public accountability [13, 14] were introduced. Quite recently, Krenn et al. further refined the security properties to also account for the signatures in analogy to the strong unforgeability of conventional signatures [40].

Invisibility of SSS, formalized by Camenisch et al. [15], prohibits that an outsider can decide which blocks are admissible, dating back to the original ideas by Ateniese et al. [3]. We extend their work, and use the aforementioned results as our starting point for the strengthened definitions. Miyazaki et al. also introduce invisibility of sanitizable signatures [46]. However, they actually address the related, but different [44], concept of redactable signatures [9, 25, 37, 50, 51].

Also, several extensions such as limiting the sanitizer to certain values [18, 26, 39, 49], SSSs which allow the signer to add new sanitizers after signing [20, 52], SSSs for multi-sanitizer and multi-signer environments [11, 14, 19], as well as sanitization of signed encrypted data [22, 29] have been considered. SSSs have also been used as a tool to make other primitives accountable [48], and to construct other primitives, such as redactable signatures [8, 44]. Also, SSSs for data-structures that are more complex than simple lists have been considered [49]. Our results carry over to the aforementioned extended settings with only minor additional adjustments. Implementations of SSSs are also presented, proving that this primitive is practical [13, 14, 43, 47].

Finally, we note that computing on signed messages is a much broader field, and refer to [1, 9, 23, 32, 33] for comprehensive overviews of other related primitives.

2 Preliminaries

Let us give our notation, assumptions, and the required building blocks, first. All formal security definitions are given in App. A.

Notation. The main security parameter is denoted by $\lambda \in \mathbb{N}$. All algorithms implicitly take 1^λ as an additional input. We write $a \leftarrow A(x)$ if a is assigned to the output of algorithm A with input x. If we use external random coins r, we use the notation $a \leftarrow A(x;r)$, where $r \in \{0,1\}^\lambda$. An algorithm is efficient if it runs in probabilistic polynomial time (ppt) in the length of its input. For the remainder of this paper, all algorithms are ppt if not explicitly mentioned otherwise. Most algorithms may return a special error symbol $\perp \notin \{0,1\}^*$, denoting an exception. If S is a set, we write $a \leftarrow S$ to denote that a is chosen uniformly at random from S. For a message $m = (m[1], m[2], \ldots, m[\ell])$, we call $m[i]$ a block, while $\ell \in \mathbb{N}$ denotes the number of blocks in a message m. For a list we require that we have a unique, injective, and efficiently reversible, encoding, mapping the list to $\{0,1\}^*$. In the definitions, we speak of a general message space $\mathcal{M}$ to be as generic as possible. For our instantiations, however, we let the message space $\mathcal{M}$ be $\{0,1\}^*$ to reduce unhelpful boilerplate notation. A function $\nu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is negligible, if it vanishes faster than every inverse polynomial, i.e., $\forall k \in \mathbb{N}$, $\exists n_0 \in \mathbb{N}$ such that $\nu(n) \leq n^{-k}$, $\forall n > n_0$. For certain security properties we require that values only have one canonical representation, e.g., a "2" is not the same as a "02", even if written as elements of $\mathbb{N}$.

Pseudo-Random Functions PRF.

Definition 1 (Pseudo-Random Functions PRF). *A pseudo-random function* PRF *consists of two algorithms* $(\mathsf{KGen}_{\mathsf{prf}}, \mathsf{Eval}_{\mathsf{prf}})$ *such that:*

$\mathsf{KGen}_{\mathsf{prf}}$**.** *The algorithm* $\mathsf{KGen}_{\mathsf{prf}}$ *outputs the secret key of the* PRF*:* $\kappa \leftarrow \mathsf{KGen}_{\mathsf{prf}}(1^\lambda)$.

$\mathsf{Eval}_{\mathsf{prf}}$**.** *The deterministic algorithm* $\mathsf{Eval}_{\mathsf{prf}}$ *gets as input the key* κ*, and the value* $x \in \{0,1\}^\lambda$*, to evaluate. It outputs the evaluated value* $v \leftarrow \mathsf{Eval}_{\mathsf{prf}}(\kappa, x)$*,* $v \in \{0,1\}^\lambda$.

Pseudo-Random Generators PRG. We assume PRGs with a constant stretching factor of 2, as this is sufficient for our setting.

Definition 2 (Pseudo-Random Number-Generators PRG). *A pseudo-random number-generator* PRG *consists of one algorithm* ($\mathsf{Eval}_{\mathsf{prg}}$) *such that:*

$\mathsf{Eval}_{\mathsf{prg}}$**.** *The deterministic algorithm* $\mathsf{Eval}_{\mathsf{prg}}$ *gets as input the value* $x \in \{0,1\}^{\lambda}$ *to evaluate. It outputs the evaluated value* $v \leftarrow \mathsf{Eval}_{\mathsf{prg}}(x)$*,* $v \in \{0,1\}^{2\lambda}$*.*

Digital Signatures Σ. Subsequently, we introduce unique and strongly unforgeable signatures.

Definition 3 (Digital Signatures Σ). *A signature scheme* Σ *is a triple* ($\mathsf{KGen}_{\Sigma}, \mathsf{Sign}_{\Sigma}, \mathsf{Verify}_{\Sigma}$) *of ppt algorithms such that:*

KGen_{Σ}**.** *The algorithm* KGen_{Σ} *outputs the public, and the corresponding private key:* $(\mathsf{sk}_{\Sigma}, \mathsf{pk}_{\Sigma}) \leftarrow \mathsf{KGen}_{\Sigma}(1^{\lambda})$.

Sign_{Σ}**.** *The algorithm* Sign_{Σ} *gets as input the* sk_{Σ}*, the message* $m \in \mathcal{M}$*, and outputs a signature* $\sigma \leftarrow \mathsf{Sign}_{\Sigma}(\mathsf{sk}_{\Sigma}, m)$.

Verify_{Σ}**.** *The deterministic algorithm* Verify_{Σ} *receives as input a public key* pk_{Σ} *a message* m*, and a signatures* σ*. It outputs a decision bit* $d \in \{\mathtt{false}, \mathtt{true}\}$*:* $d \leftarrow \mathsf{Verify}_{\Sigma}(\mathsf{pk}_{\Sigma}, m, \sigma)$.

Definition 4 (Secure Digital Signatures). *A signature scheme* Σ *is secure, if it is correct, strongly unforgeable, and unique.*

A concrete instantiation satisfying Definition 4 is RSA-FDH, where the signer also proves the well-formedness of the public key, i.e., that it defines a permutation, and is not lossy [38]. This can, e.g., be achieved by requiring a prime public exponent e larger than the modulus n, and a verifier also checks that $\sigma \in \mathbb{Z}_n^*$.

Labeled Public-Key Encryption Schemes Π. Public-key encryption with labels allows to encrypt a message m using a given public key pk so that the resulting ciphertext can be decrypted using the corresponding secret key sk [17], and some, potentially public, label ϑ:

Definition 5 (Labeled Public-Key Encryption Schemes). *A labeled public-key encryption scheme* Π *is a triple* ($\mathsf{KGen}_{\Pi}, \mathsf{Enc}_{\Pi}, \mathsf{Dec}_{\Pi}$) *of ppt algorithms such that:*

KGen_{Π}**.** *The algorithm* KGen_{Π} *outputs the private, and public, keys of the scheme:* $(\mathsf{sk}_{\Pi}, \mathsf{pk}_{\Pi}) \leftarrow \mathsf{KGen}_{\Pi}(1^{\lambda})$.

Enc_{Π}**.** *The algorithm* Enc_{Π} *gets as input the public key* pk_{Π}*, the message* $m \in \mathcal{M}$*, and some label* $\vartheta \in \{0,1\}^*$*. It outputs a ciphertext* c*:* $c \leftarrow \mathsf{Enc}_{\Pi}(\mathsf{pk}_{\Pi}, m, \vartheta)$.

Dec_{Π}**.** *The deterministic algorithm* Dec_{Π} *on input a private key* sk_{Π}*, a ciphertext* c*, and some label* ϑ *outputs a message* $m \in \mathcal{M} \cup \{\perp\}$*:* $m \leftarrow \mathsf{Dec}_{\Pi}(\mathsf{sk}_{\Pi}, c, \vartheta)$.

Definition 6 (Secure Labeled Public-Key Encryption Schemes). *A labeled public-key encryption scheme* Π *is secure, if it is correct, and IND-CCA2-secure.*

Chameleon-Hashes. The given framework is based upon the work done by Camenisch et al. [15].

Definition 7. *A chameleon-hash* CH *is a tuple of five ppt algorithms* (CHPGen, CHKGen, CHash, CHCheck, CHAdapt)*, such that:*

CHPGen**.** *The algorithm* CHPGen *outputs public parameters of the scheme:* $\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{CHPGen}(1^{\lambda})$*. For brevity, we assume that* $\mathsf{pp}_{\mathsf{ch}}$ *is implicit input to all other algorithms.*

CHKGen**.** *The algorithm* CHKGen *given the public parameters* $\mathsf{pp}_{\mathsf{ch}}$ *outputs the private, and public, keys of the scheme:* $(\mathsf{sk}_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{ch}}) \leftarrow \mathsf{CHKGen}(\mathsf{pp}_{\mathsf{ch}})$.

Experiment $\mathsf{Indistinguishability}^{\mathsf{CH}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{CHPGen}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{ch}}) \leftarrow \mathsf{CHKGen}(\mathsf{pp}_{\mathsf{ch}})$
$b \leftarrow \{0,1\}$
$a \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt}(\mathsf{sk}_{\mathsf{ch}},\cdot,\cdot,b),\mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{ch}})$
where oracle $\mathsf{HashOrAdapt}$ on input $\mathsf{sk}_{\mathsf{ch}}, m, m', b$:
$(h, r) \leftarrow \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m')$
$(h', r') \leftarrow \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m)$
$r'' \leftarrow \mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, m, m', r', h')$
If $r = \bot \ \vee \ r'' = \bot$, return $\bot$
if $b = 0$:
return (h, r)
if $b = 1$:
return (h', r'')
return 1, if $a = b$
return 0

Fig. 1: CH Indistinguishability

Experiment $\mathsf{CollRes}^{\mathsf{CH}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{CHPGen}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{ch}}) \leftarrow \mathsf{CHKGen}(\mathsf{pp}_{\mathsf{ch}})$
$\mathcal{Q} \leftarrow \emptyset$
$(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{CHAdapt}'(\mathsf{sk}_{\mathsf{ch}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{ch}})$
where oracle $\mathsf{CHAdapt}'$ on input $\mathsf{sk}_{\mathsf{ch}}, m, m', r, h$:
return $\bot$, if $\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m, r, h) \neq \mathtt{true}$
$r' \leftarrow \mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, m, m', r, h)$
If $r' = \bot$, return $\bot$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m, m'\}$
return r'
return 1, if $\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m^*, r^*, h^*) = \mathtt{true} \wedge$
$\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m'^*, r'^*, h^*) = \mathtt{true} \wedge$
$m'^* \notin \mathcal{Q} \ \wedge \ m^* \neq m'^*$
return 0

Fig. 2: CH Collision Resistance

CHash. *The algorithm* CHash *gets as input the public key* $\mathsf{pk}_{\mathsf{ch}}$*, and a message* m *to hash. It outputs a hash* h*, and some randomness* r*:* $(h, r) \leftarrow \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m)$.[8]

CHCheck. *The deterministic algorithm* CHCheck *gets as input the public key* $\mathsf{pk}_{\mathsf{ch}}$*, a message* m*, randomness* r*, and a hash* h*. It outputs a decision* $d \in \{\mathtt{false}, \mathtt{true}\}$ *indicating whether the hash* h *is valid:* $d \leftarrow \mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m, r, h)$.

CHAdapt. *The algorithm* CHAdapt *on input of secret key* $\mathsf{sk}_{\mathsf{ch}}$*, the message* m*, the randomness* r*, hash* h*, and a new message* m' *outputs new randomness* r'*:* $r' \leftarrow \mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, m, m', r, h)$.

Correctness. For a CH we require the correctness property to hold. In particular, we require that for all $\lambda \in \mathbb{N}$, for all $\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{CHPGen}(1^\lambda)$, for all $(\mathsf{sk}_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{ch}}) \leftarrow \mathsf{CHKGen}(\mathsf{pp}_{\mathsf{ch}})$, for all $m \in \mathcal{M}$, for all $(h, r) \leftarrow \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m)$, for all $m' \in \mathcal{M}$, we have for all for all $r' \leftarrow \mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, m, m', r, h)$, that $\mathtt{true} = \mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m, r, h) = \mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m', r', h)$. This definition captures perfect correctness.

Indistinguishability. Indistinguishability requires that the randomness r does not reveal if it was obtained through CHash or CHAdapt. The messages are chosen by the adversary.

Definition 8 (Indistinguishability). *A chameleon-hash* CH *is indistinguishable, if for any ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\left|\Pr[\mathsf{Indistinguishability}^{\mathsf{CH}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$*. The corresponding experiment is depicted in Fig. 1.*

Collision Resistance. Collision resistance says, that even if an adversary has access to an adapt oracle, it cannot find any collisions for messages other than the ones queried to the adapt oracle. Note, this definition is stronger than key-exposure freeness [5].

Definition 9 (Collision-Resistance). *A chameleon-hash* CH *is collision-resistant, if for any ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{CollRes}^{\mathsf{CH}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$*. The corresponding experiment is depicted in Fig. 2.*

Uniqueness. Uniqueness requires that it is hard to come up with two different randomness values for the same message m^* such that the hashes are equal, for the same adversarially chosen pk^*.

[8] The randomness r is also sometimes called "check value" [4].

Experiment $\mathsf{Uniqueness}^{\mathsf{CH}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{CHPGen}(1^\lambda)$
$(\mathsf{pk}^*, m^*, r^*, r'^*, h^*) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{ch}})$
return 1, if $\mathsf{CHCheck}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{CHCheck}(\mathsf{pk}^*, m^*, r'^*, h^*) = \mathtt{true} \ \wedge \ r^* \neq r'^*$
return 0

Fig. 3: CH Uniqueness

$\mathsf{CHPGen}(1^\lambda)$: Call RSAKGen with the restriction $e > n'$, and e prime. Return e.
$\mathsf{CHKGen}(\mathsf{pp}_{\mathsf{ch}})$: Generate p, q using $\mathsf{RSAKGen}(1^\lambda)$. Let $n = pq$. Compute d such that $ed \equiv 1 \bmod \varphi(n)$. Return $(\mathsf{sk}_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{ch}}) = (d, n)$.
$\mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, m)$: Draw $r \leftarrow \mathbb{Z}_n^*$. If external random coins r' is used, one can use the random oracle, i.e., $r \leftarrow \mathcal{H}_n(r')$. Let $h \leftarrow \mathcal{H}_n(m)r^e \bmod n$. Return (h, r).
$\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m, r, h')$: If $r \notin \mathbb{Z}_n^*$, return $\mathtt{false}$. Let $h \leftarrow \mathcal{H}_n(m)r^e \bmod n$. Return $\mathtt{true}$, if $h = h'$, and $\mathtt{false}$ otherwise.
$\mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, m, m', r, h)$: If $\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, m, r, h) = \mathtt{false}$, return $\perp$. If $m = m'$, return r. Let $g \leftarrow \mathcal{H}_n(m)$, $y \leftarrow gr^e \bmod n$, and $g' \leftarrow \mathcal{H}_n(m')$. Return $r' \leftarrow (y(g'^{-1}))^d \bmod n$.

Construction 1: Secure CH

Definition 10 (Uniqueness). *A chameleon-hash* CH *is unique, if for any ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{Uniqueness}^{\mathsf{CH}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 3.*

Definition 11 (Secure Chameleon-Hashes). *We call a chameleon-hash* CH *secure, if it is correct, indistinguishable, collision-resistant, and unique.*

RSA Instance Generator. Let $(n, p, q, e, d) \leftarrow \mathsf{RSAKGen}(1^\lambda)$ be an instance generator which returns an RSA modulus $n = pq$, where p and q are distinct primes, $e > n'$ an integer co-prime to $\varphi(n)$, and $de \equiv 1 \bmod \varphi(n)$. Here, n' is the largest RSA modulus possible w.r.t. λ. It is assumed that e is prime and chosen independently of p and q, while d is calculated from e, and not vice versa.

The Chameleon-Hash by Camenisch et al. [15]. Next, as Construction 1, we restate the construction by Camenisch et al. [15], which is secure, if the one-more RSA-Assumption [6] holds. Now, let $\mathsf{CH} := (\mathsf{CHPGen}, \mathsf{CHKGen}, \mathsf{CHash}, \mathsf{CHCheck}, \mathsf{CHAdapt})$ as defined in Construction 1. $\mathcal{H}_n : \{0,1\}^* \to \mathbb{Z}_n^*$, with $n \in \mathbb{N}$, denotes a random oracle. Each n is implicitly required to have λ bits. This is not explicitly checked in the algorithms.

3 Stronger Invisible Sanitizable Signatures

We now present our framework for strongly invisible sanitizable signatures, along with the strengthened security model, and a provably secure construction based on only standard primitives.

3.1 The Framework for Sanitizable Signature Schemes

Subsequently, we introduce the framework for SSSs. The definitions are essentially the ones given by Camenisch et al. [15], which are itself based on existing work [10, 13, 14, 3, 34, 40]. However, due to our goals, we need to re-define the security experiments. Like Camenisch et al. [15], we do not consider "non-interactive public accountability" [13, 14, 36], which allows a third party to decide which party is accountable, instead transparency is achieved, which is mutually exclusive to this property. However, it remains elegantly easy to achieve, e.g., by signing the signature again [13, 15].

For brevity, we now set some additional notation. This notation is based on existing definitions, making reading more comfortable [10, 15]. The variable ADM contains the set of indices of the modifiable blocks,

as well as ℓ denoting the total number of blocks in the message m. We write $\text{ADM}(m) = \texttt{true}$, if ADM is valid w.r.t. m, i.e., ADM contains the correct ℓ and all indices are in m. For example, let $\text{ADM} = (\{1, 2, 4\}, 4)$. Then, m must contain four blocks, and all but the third are admissible. If we write $m_i \in \text{ADM}$, we mean that m_i is admissible. MOD is a set containing pairs $(i, m[i]')$ for those blocks that are modified, meaning that $m[i]$ is replaced with $m[i]'$. We write $\text{MOD}(\text{ADM}) = \texttt{true}$, if MOD is valid w.r.t. ADM, meaning that the indices to be modified are contained in ADM. To allow for a compact presentation of our construction, we write $[X]_{n,m}$, with $n \leq m$, for the vector $(X_n, X_{n+1}, X_{n+2}, \ldots, X_{m-1}, X_m)$.

Definition 12 (Sanitizable Signatures). *A sanitizable signature scheme* SSS *consists of the ppt algorithms* $(\mathsf{SSSPGen}, \mathsf{KGen}_{\mathsf{sig}}, \mathsf{KGen}_{\mathsf{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *such that:*

$\mathsf{SSSPGen}$**.** *The algorithm* $\mathsf{SSSPGen}$*, on input security parameter* λ*, generates the public parameters:* $\mathsf{pp}_{\mathsf{sss}} \leftarrow \mathsf{SSSPGen}(1^\lambda)$*. We assume that* $\mathsf{pp}_{\mathsf{sss}}$ *is implicitly input to all other algorithms.*

$\mathsf{KGen}_{\mathsf{sig}}$**.** *The algorithm* $\mathsf{KGen}_{\mathsf{sig}}$ *takes the public parameters* $\mathsf{pp}_{\mathsf{sss}}$*, and returns the signer's private key and the corresponding public key:* $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{sss}})$*.*

$\mathsf{KGen}_{\mathsf{san}}$**.** *The algorithm* $\mathsf{KGen}_{\mathsf{san}}$ *takes the public parameters* $\mathsf{pp}_{\mathsf{sss}}$*, and returns the sanitizer's private key as well as the corresponding public key:* $(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) \leftarrow \mathsf{KGen}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{sss}})$*.*

Sign**.** *The algorithm* Sign *takes as input a message* m*,* $\mathsf{sk}_{\mathsf{sig}}$*,* $\mathsf{pk}_{\mathsf{san}}$*, as well as a description* ADM *of the admissible blocks. If* $\text{ADM}(m) = \texttt{false}$*, this algorithm returns* $\perp$*. It outputs a signature* $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \text{ADM})$*.*

Sanit**.** *The algorithm* Sanit *takes a message* m*, modification instruction* MOD*, a signature* σ*,* $\mathsf{pk}_{\mathsf{sig}}$*, and* $\mathsf{sk}_{\mathsf{san}}$*. It outputs* m' *together with* σ'*:* $(m', \sigma') \leftarrow \mathsf{Sanit}(m, \text{MOD}, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}})$ *where* $m' \leftarrow \text{MOD}(m)$ *is message* m *modified according to the modification instruction* MOD*.*

Verify**.** *The algorithm* Verify *takes as input the signature* σ *for a message* m *w.r.t. the public keys* $\mathsf{pk}_{\mathsf{sig}}$*, and* $\mathsf{pk}_{\mathsf{san}}$*. It outputs a decision* $d \in \{\texttt{true}, \texttt{false}\}$*:* $d \leftarrow \mathsf{Verify}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$*.*

Proof**.** *The algorithm* Proof *takes as input* $\mathsf{sk}_{\mathsf{sig}}$*, a message* m*, a signature* σ*, a set of polynomially many additional message/signature pairs* $\{(m_i, \sigma_i)\}$*, and* $\mathsf{pk}_{\mathsf{san}}$*. It outputs a string* $\pi \in \{0,1\}^*$ *which can be used by the* Judge *to decide which party is accountable given a message/signature pair* (m, σ)*:* $\pi \leftarrow \mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \mathsf{pk}_{\mathsf{san}})$*.*

Judge**.** *The algorithm* Judge *takes as input a message* m*, a signature* σ*,* $\mathsf{pk}_{\mathsf{sig}}$*,* $\mathsf{pk}_{\mathsf{san}}$*, as well as a proof* π*. Note, this means that once a proof* π *is generated, the accountable party can be derived by anyone for that message/signature pair* (m, σ)*. It outputs a decision* $d \in \{\mathsf{Sig}, \mathsf{San}\}$*, indicating whether the message/signature pair has been created by the signer, or the sanitizer:* $d \leftarrow \mathsf{Judge}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \pi)$*.*

Correctness of Sanitizable Signature Schemes. The usual correctness requirements must hold, i.e., every signed and sanitized message/signature pair should verify, while a honestly generated proof on a honestly generated message/signature pair points to the correct accountable party. We refer to Brzuska et al. [10] for a formal definition, which straightforwardly extends to this framework.

3.2 Security of Sanitizable Signature Schemes

Next, we introduce the security model, based on the work done by Camenisch et al. [15], but extended to account for our new insights. In other words, we strengthen their invisibility notion, and achieve *strong* signer-accountability [40].

Unforgeability. This definition requires that an adversary $\mathcal{A}$, not having any secret keys, is not able to produce *any* valid signature σ^* which it has not seen, even if $\mathcal{A}$ has full oracle access.

Definition 13 (Unforgeability). *An* SSS *is unforgeable, if for any ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{Unforgeability}_{\mathcal{A}}^{\mathsf{SSS}}(\lambda) = 1] \leq \nu(\lambda)$*, where the corresponding experiment is defined in Fig. 4.*

Experiment $\mathsf{Unforgeability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{sss}} \leftarrow \mathsf{SSSPGen}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{sss}})$
$(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) \leftarrow \mathsf{KGen}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{sss}})$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{san}})}_{\mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$
for $i = 1, 2, \ldots, q$ let $(m_i, \mathsf{pk}_{\mathsf{san},i}, \mathrm{ADM}_i)$ and σ_i
index the queries/answers to/from Sign
for $j = 1, 2, \ldots, q'$ let $(m_j, \sigma_j, \mathsf{pk}_{\mathsf{sig},j}, \mathrm{MOD}_j)$ and (m'_j, σ'_j)
index the queries/answers to/from Sanit
return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) = \mathtt{true} \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}_{\mathsf{san}}, m^*, \sigma^*) \neq (\mathsf{pk}_{\mathsf{san},i}, m_i, \sigma_i) \wedge$
$\forall j \in \{1, 2, \ldots, q'\} : (\mathsf{pk}_{\mathsf{sig}}, m^*, \sigma^*) \neq (\mathsf{pk}_{\mathsf{sig},j}, m'_j, \sigma'_j)$
return 0

Fig. 4: SSS Unforgeability

Experiment $\mathsf{Immutability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{sss}} \leftarrow \mathsf{SSSPGen}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{sss}})$
$(m^*, \sigma^*, \mathsf{pk}^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot),\mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{sig}})$
for $i = 1, 2, \ldots, q$ let $(m_i, \mathsf{pk}_{\mathsf{san},i}, \mathrm{ADM}_i)$
index the queries to Sign
return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*) = \mathtt{true} \wedge$
$(\forall i \in \{1, 2, \ldots, q\} : \mathsf{pk}^* \neq \mathsf{pk}_{\mathsf{san},i} \vee$
$m^* \notin \{\mathrm{MOD}(m_i) \mid \mathrm{MOD} \text{ with } \mathrm{MOD}(\mathrm{ADM}_i) = \mathtt{true}\})$
return 0

Fig. 5: SSS Immutability

Immutability. A sanitizer must only be able to sanitize the admissible blocks defined by ADM. This also prohibits deleting, or appending, blocks from a given message m. The adversary is given full oracle access, while it is also allowed to generate the sanitizer key pair itself.

Definition 14 (Immutability). *An* SSS *is immutable, if for any ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{Immutability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 5.*

Privacy. The notion of privacy is related to the indistinguishability of ciphertexts. The adversary is allowed to input two messages with the same ADM which are sanitized to the exact same message. The adversary then has to decide which of the input messages was used to generate the sanitized one. The adversary receives full adaptive oracle access. Note, the adversary never receives un-sanitized signatures from the $\mathsf{LoRSanit}$ oracle, and can thus never query them to the Proof oracle. Thus, no proofs for those signatures are generated.

Definition 15 (Privacy). *An* SSS *is private, if for any ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\left|\Pr[\mathsf{Privacy}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 6.*

Transparency. Transparency guarantees that the accountable party of a message m remains anonymous. This is important if discrimination may follow [3, 10]. In a nutshell, the adversary has to decide whether it sees a freshly computed signature, or a sanitized one. The adversary has full (but proof-restricted) adaptive oracle access. We stress that we use the proof-restricted version by Camenisch et al. [15].

Definition 16 ((Proof-Restricted) Transparency). *An* SSS *is proof-restricted transparent, if for any ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\left|\Pr[\mathsf{Transparency}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 7.*

From now on, we use the term "transparency", even if we mean proof-restricted transparency.

Strong Signer-Accountability. For strong signer-accountability, a signer must not be able to blame a sanitizer if the sanitizer is actually not responsible for a given message/signature pair. Hence, the adversary $\mathcal{A}$ has to generate a proof π^* which makes Judge to decide that the sanitizer is accountable, if it is not for a message/signature pair (m^*, σ^*) output by $\mathcal{A}$. Here, the adversary gains access to all oracles related to sanitizing. This definition *does* take the signature into account.

Definition 17 (Strong Signer-Accountability). *An* SSS *is strongly signer-accountable, if for any ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{SSig\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$, *where the experiment is defined in Fig. 8.*

Experiment $\mathsf{Privacy}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{sss}} \leftarrow \mathsf{SSSPGen}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{sss}})$
$(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) \leftarrow \mathsf{KGen}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{sss}})$
$b \leftarrow \{0,1\}$
$a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{san}}),\mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot)}_{\mathsf{LoRSanit}(\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{sig}},\mathsf{sk}_{\mathsf{san}},b)}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$
where oracle $\mathsf{LoRSanit}$ on input of
$m_0, m_1, \mathrm{MOD}_0, \mathrm{MOD}_1, \mathrm{ADM}, \mathsf{sk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}}, b$
return $\perp$, if $\mathrm{MOD}_0(m_0) \neq \mathrm{MOD}_1(m_1) \vee$
$\mathrm{ADM}(m_0) \neq \mathrm{ADM}(m_1)$
let $\sigma \leftarrow \mathsf{Sign}(m_b, \mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \mathrm{ADM})$
return $(m', \sigma') \leftarrow \mathsf{Sanit}(m_b, \mathrm{MOD}_b, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}})$
return 1, if $a = b$
return 0

Fig. 6: SSS Privacy

Experiment $\mathsf{Transparency}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{sss}} \leftarrow \mathsf{SSSPGen}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{sss}})$
$(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) \leftarrow \mathsf{KGen}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{sss}})$
$b \leftarrow \{0,1\}$
$\mathcal{Q} \leftarrow \emptyset$
$a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{san}}),\mathsf{Proof}'(\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot)}_{\mathsf{Sanit/Sign}(\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{sig}},\mathsf{sk}_{\mathsf{san}},b)}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$
where oracle Proof' on input of
$\mathsf{sk}_{\mathsf{sig}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \mathsf{pk}'_{\mathsf{san}}, b$:
return $\perp$, if $\mathsf{pk}'_{\mathsf{san}} = \mathsf{pk}_{\mathsf{san}} \ \wedge \ (m, \sigma) \in \mathcal{Q}$
return $\mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \mathsf{pk}'_{\mathsf{san}})$
where oracle $\mathsf{Sanit/Sign}$ on input of
$m, \mathrm{MOD}, \mathrm{ADM}, \mathsf{sk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}}, b$:
$\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \mathrm{ADM})$
$(m', \sigma') \leftarrow \mathsf{Sanit}(m, \mathrm{MOD}, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}})$
if $b = 1$:
$\sigma' \leftarrow \mathsf{Sign}(m', \mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \mathrm{ADM})$
If $\sigma' \neq \perp$, set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m', \sigma')\}$
return (m', σ')
return 1, if $a = b$
return 0

Fig. 7: SSS (Proof-Restricted) Transparency

Experiment $\mathsf{SSig\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{sss}} \leftarrow \mathsf{SSSPGen}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) \leftarrow \mathsf{KGen}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{sss}})$
$(\mathsf{pk}^*, \pi^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{san}})}(\mathsf{pk}_{\mathsf{san}})$
for $i = 1, 2, \ldots, q$ let (m'_i, σ'_i) and $(m_i, \mathrm{MOD}_i, \sigma_i, \mathsf{pk}_{\mathsf{sig},i})$
index the answers/queries from/to Sanit
return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk}_{\mathsf{san}}) = \mathtt{true} \ \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^*, m^*, \sigma^*) \neq (\mathsf{pk}_{\mathsf{sig},i}, m'_i, \sigma'_i) \ \wedge$
$\mathsf{Judge}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk}_{\mathsf{san}}, \pi^*) = \mathsf{San}$
return 0

Fig. 8: SSS Strong Signer-Accountability

Experiment $\mathsf{San\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{sss}} \leftarrow \mathsf{SSSPGen}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{sss}})$
$(m^*, \sigma^*, \mathsf{pk}^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot),\mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{sig}})$
for $i = 1, 2, \ldots, q$ let $(m_i, \mathrm{ADM}_i, \mathsf{pk}_{\mathsf{san},i})$ and σ_i
index the queries/answers to/from Sign
$\pi \leftarrow \mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}}, m^*, \sigma^*, \{(m_i, \sigma_i) \mid 0 < i \leq q\}, \mathsf{pk}^*)$
return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*) = \mathtt{true} \ \wedge$
$\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^*, m^*, \sigma^*) \neq (\mathsf{pk}_{\mathsf{san},i}, m_i, \sigma_i) \ \wedge$
$\mathsf{Judge}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*, \pi) = \mathsf{Sig}$
return 0

Fig. 9: SSS Sanitizer Accountability

Sanitizer-Accountability. Sanitizer-accountability requires that the sanitizer cannot blame the signer for a message/signature pair not created by the signer. In particular, the adversary has to make Proof generate a proof π which makes Judge decide that for a given (m^*, σ^*) generated by $\mathcal{A}$ the signer is accountable, while it is not. Thus, the adversary $\mathcal{A}$ gains access to all signer-related oracles.

Definition 18 (Sanitizer-Accountability). *An* SSS *is sanitizer-accountable, if for any ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{San\text{-}Accountability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$, *where the experiment is defined in Fig. 9.*

As Camenisch et al. [15], we do not consider unlinkability [12, 14, 30, 41] in our construction, as it seems to be very hard to achieve with the underlying construction paradigm.

3.3 Strong Invisibility of SSSs

Subsequently, we introduce the property of *strong* invisibility. The definition is derived from the one given by Camenisch et al. [15], but allows queries to the sanitization oracle with all adversarially chosen public keys.

In a nutshell, the adversary can query an LoRADM oracle which either makes ADM_0 or ADM_1 admissible in the final signature. Of course, the adversary has to be restricted to $\mathrm{ADM}_0 \cap \mathrm{ADM}_1$ for sanitization requests

Experiment $\mathsf{SInvisibility}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp}_{\mathsf{sss}} \leftarrow \mathsf{SSSPGen}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{sss}})$
$(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) \leftarrow \mathsf{KGen}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{sss}})$
$b \leftarrow \{0,1\}$
$\mathcal{Q} \leftarrow \emptyset$
$a \leftarrow \mathcal{A}^{\mathsf{Sanit}'(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{san}}),\mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot),\mathsf{LoRADM}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{sig}},b)}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$
where oracle LoRADM on input of $m, \mathrm{ADM}_0, \mathrm{ADM}_1, \mathsf{pk}'_{\mathsf{san}}, \mathsf{sk}_{\mathsf{sig}}, b$:
return $\bot$, if $\mathrm{ADM}_0(m) \neq \mathrm{ADM}_1(m)$
return $\bot$, if $\mathsf{pk}_{\mathsf{san}} \neq \mathsf{pk}'_{\mathsf{san}} \ \wedge \ \mathrm{ADM}_0 \neq \mathrm{ADM}_1$
let $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \mathrm{ADM}_b)$
if $\mathsf{pk}'_{\mathsf{san}} = \mathsf{pk}_{\mathsf{san}}$, let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m, \sigma, \mathrm{ADM}_0 \cap \mathrm{ADM}_1)\}$
return σ
where oracle Sanit' on input of $m, \mathrm{MOD}, \sigma, \mathsf{pk}'_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}}$:
return $\bot$, if $\mathsf{pk}'_{\mathsf{sig}} = \mathsf{pk}_{\mathsf{sig}} \wedge \nexists (m, \sigma, \mathrm{ADM}) \in \mathcal{Q} : \mathrm{MOD}(\mathrm{ADM}) = \texttt{true}$
let $(m', \sigma') \leftarrow \mathsf{Sanit}(m, \mathrm{MOD}, \sigma, \mathsf{pk}'_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}})$
if $\mathsf{pk}'_{\mathsf{sig}} = \mathsf{pk}_{\mathsf{sig}} \wedge \exists (m, \sigma, \mathrm{ADM}') \in \mathcal{Q} : \mathrm{MOD}(\mathrm{ADM}') = \texttt{true}$,
let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m', \sigma', \mathrm{ADM}')\}$
return (m', σ')
return 1, if $a = b$
return 0

Fig. 10: SSS Strong Invisibility

for signatures originating from those created by LoRADM, and their derivatives, to avoid trivial attacks with the challenge key. However, compared to the original definition, we do not restrict that the signer public key is the challenge one. Moreover, as in the original definition, the sign oracle can be simulated by querying the LoRADM oracle with $\mathrm{ADM}_0 = \mathrm{ADM}_1$.

Definition 19 (Strong Invisibility). *An SSS is strongly invisible, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that $\left|\Pr[\mathsf{SInvisibility}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$, where the corresponding experiment is defined in Fig. 10.*

Clearly, our definition implies the invisibility definition by Camenisch et al. [15], and strong invisibility is not implied by any other property.

Definition 20 (Secure SSS). *We call an SSS secure, if it is correct, private, unforgeable, immutable, sanitizer-accountable, strongly signer-accountable, and strongly invisible.*

As mentioned before, we do not consider non-interactive public accountability, unlinkability, or transparency, as essential requirements, as it depends on the concrete use-case whether these properties are required.

3.4 Construction

Our construction is similar to the one by Camenisch et al. [15][9] but contains several improvements. In their paradigm, each block is protected by a chameleon-hash with ephemeral trapdoors under the sanitizer's key, while the hash values are signed by the signer. Then the ephemeral trapdoors for the modifiable blocks are revealed to the sanitizer, who can modify those blocks by computing collisions. Our trick is to mimic chameleon hashes with ephemeral trapdoors by generating a *fresh* chameleon hash key pair for each block, while only the overall message is protected by a chameleon hash under the sanitizer's key. Then we give the secret keys $\mathsf{sk}^i_{\mathsf{ch}}$ for which the respective block $m[i]$ is admissible to the sanitizer while the public keys $\mathsf{pk}^i_{\mathsf{ch}}$ are included in the signature by the signer. To hide whether a given block is sanitizable, each $\mathsf{sk}^i_{\mathsf{ch}}$ is encrypted;

[9] Which, in turn, is based on prior work [10, 34, 43].

a sanitizable block contains the real $\mathsf{sk}_{\mathsf{ch}}^{i}$, while a non-admissible block encrypts a 0 (0 is assumed to be an invalid $\mathsf{sk}_{\mathsf{ch}}^{i}$). To prohibit the re-use of ciphertexts from different $\mathsf{pk}_{\mathsf{sig}}$s, which is exactly the thin line between invisibility and strong invisibility, the signer also needs to put its public key $\mathsf{pk}_{\mathsf{sig}}$ into the label for each ciphertext. As we show in the proof, this then allows to simulate decryption for all requests when using labeled CCA2-secure encryption. To achieve accountability, as Camenisch et al. [15], we generate additional "tags" for a chameleon-hash (which binds everything together) in a special way, i.e., using PRFs and PRGs. This idea can essentially be tracked back to Brzuska et al. [10]. To achieve strong signer-accountability, we need to resort to *unique* signature schemes, and generate the randomness of the chameleon hash (one can use the one given in Sect. 2) in a special way. Namely, we use the unique signature scheme to sign a random value. The resulting signature is hashed, and used as a randomness source for the outer chameleon-hash. To maintain transparency, the signature is encrypted to the sanitizer, who verifies that the signature is correct upon every sanitization, and does not produce sanitized signatures otherwise. This is necessary, as the definition of CH collision-resistance (see Fig. 2) does not rule out that the adversary can find new colliding hashes for already seen collisions. Our trick prohibits such attacks. In more detail, the proof π needs to also contain the randomness used to generate the chameleon-hash, which is exactly the hashed signature on a public random value. The new construction is given in Construction 2, where $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ denotes a random oracle [7].

Theorem 1 (proven in App. B). *If Π, Σ, and* CH*, are secure, while* PRF*, and* PRG*, are pseudo-random, Construction 2 is a secure, and transparent,* SSS*.*

4 Evaluation

To demonstrate the practicality of our scheme, we have implemented our construction in Java. The chameleon-hash CH we implemented is the one presented in Sect. 2. All RSA-moduli for Σ and CH have a fixed bit-length of 2,048 Bit (with balanced primes). Likewise, each e generated (one for Σ, and one for CH) has 2,050 Bit. Σ is a standard RSA-FDH implementation, with the required constraints. We have fixed the output size of $\mathcal{H}$, PRF, and PRG, to 512 and $2 \cdot 512$, respectively, as these sizes turned out to yield the best performance when using 2,048 Bit moduli. $\mathcal{H}$, and PRF, were implemented using SHA-512. $\mathcal{H}_n$ is SHA-512 in counter-mode [7], similar to what is used in existing implementations [16, 31]. Π was implemented using the IND-CCA2-secure version of RSA-OAEP (2,048 Bit modulus), paired with a symmetric encrypt-then-MAC cipher-suite (AES/3DES-CBC-MAC). These implementations were taken from the SCAPI-framework [28]. They use 128 Bit encryption keys, and 112 Bit MAC keys.

The measurements were performed on a Lenovo W530 with an Intel i7-3470QM@2.70Ghz, and 16GiB of RAM. No performance optimization such as CRT were implemented, and only a single thread does the computations. This was done to see the actual lower bound of our construction, i.e., any additional optimization helps. We evaluated our implementation with 32 blocks, wheres 50% were marked as admissible. For sanitization, 50% of the admissible blocks were sanitized, i.e., 8 blocks. We omitted proof generation, and the judge, as they are simple database look-ups, paired with comparisons, and depend on the number of signatures generated. The overall results are depicted in Fig. 11a, and Tab. 11b and are based on 200 runs. Parameter generation is also omitted, as this is a one-time setup, i.e., drawing a random prime with 2,050 Bit. As demonstrated, signing is the most expensive operation. The lion's share is finding suitable primes; the exponentiations within the algorithms only have a negligible overhead, as seen by the runtime of sanitization, and verification. A practical optimization would therefore be to generate the generated keys in advance, when no other computation is done, or even in parallel. However, even without these optimization the runtime can be considered practical, also with decent security parameters, and rather expensive RSA-based primitives.

$\mathsf{SSSPGen}(1^\lambda)$: Let $\mathsf{pp}_{\mathsf{ch}} \leftarrow \mathsf{CHPGen}(1^\lambda)$. Return $\mathsf{pp}_{\mathsf{sss}} = \mathsf{pp}_{\mathsf{ch}}$.
$\mathsf{KGen}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{sss}})$: Let $(\mathsf{pk}_\Sigma, \mathsf{sk}_\Sigma) \leftarrow \mathsf{KGen}_\Sigma(1^\lambda)$, $\kappa \leftarrow \mathsf{KGen}_{\mathsf{prf}}(1^\lambda)$, and return $((\kappa, \mathsf{sk}_\Sigma), \mathsf{pk}_\Sigma)$.
$\mathsf{KGen}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{sss}})$: Let $(\mathsf{sk}_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{ch}}) \leftarrow \mathsf{CHKGen}(\mathsf{pp}_{\mathsf{sss}})$, $(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) \leftarrow \mathsf{KGen}_\Pi(1^\lambda)$, and return $((\mathsf{sk}_{\mathsf{ch}}, \mathsf{sk}_\Pi), (\mathsf{pk}_{\mathsf{ch}}, \mathsf{pk}_\Pi))$.

$\mathsf{Sign}(m, \mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \mathrm{ADM})$: If $\mathrm{ADM}(m) \neq \mathtt{true}$, return $\perp$. Let $x_0 \leftarrow \{0,1\}^\lambda$, $x_0' \leftarrow \mathsf{Eval}_{\mathsf{prf}}(\kappa, x_0)$, $\tau \leftarrow \mathsf{Eval}_{\mathsf{prg}}(x_0')$, $x_1 \leftarrow \{0,1\}^\lambda$. Further, let

$$\forall\, i \in \{1,\ldots,\ell\} \;:\; (\mathsf{sk}^i_{\mathsf{ch}}, \mathsf{pk}^i_{\mathsf{ch}}) \leftarrow \mathsf{CHKGen}(\mathsf{pp}_{\mathsf{ch}}),\; (h_i, r_i) \leftarrow \mathsf{CHash}(\mathsf{pk}^i_{\mathsf{ch}}, (i, m[i], \mathsf{pk}_{\mathsf{sig}})),$$
$$\forall\, j \notin \mathrm{ADM} \;:\; \mathsf{sk}^j_{\mathsf{ch}} \leftarrow 0, \text{ and } \forall\, i \in \{1,\ldots,\ell\} \;:\; c_i \leftarrow \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, \mathsf{sk}^i_{\mathsf{ch}}, \mathsf{pk}_{\mathsf{sig}}).$$

Return $\sigma = (\sigma', x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, [r]_{0,\ell}, \tau, c_h, [c]_{1,\ell}, [h]_{0,\ell})$, where

$$\sigma_h \leftarrow \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, x_1), c_h \leftarrow \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, (m, \sigma_h, [r]_{1,\ell}, \tau), \mathsf{pk}_{\mathsf{sig}}), t \leftarrow \mathcal{H}(\sigma_h, x_1, \mathsf{pk}_{\mathsf{sig}}),$$
$$(h_0, r_0) \leftarrow \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, (0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}); t),$$
$$\sigma' \leftarrow \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, (x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, [h]_{0,\ell}, [c]_{1,\ell}, \mathsf{pk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{sig}}, \ell))$$

$\mathsf{Verify}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$: Return $\mathtt{true}$ if all of the following checks hold, and $\mathtt{false}$ otherwise:

$$\forall i \in \{1,\ldots,\ell\} \;:\; \mathsf{CHCheck}(\mathsf{pk}^i_{\mathsf{ch}}, (i, m[i], \mathsf{pk}_{\mathsf{sig}}), r_i, h_i) = \mathtt{true} \;\wedge$$
$$\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, (0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}), r_0, h_0) = \mathtt{true} \;\wedge$$
$$\mathsf{Verify}_\Sigma(\mathsf{pk}_\Sigma, (x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, [h]_{0,\ell}, c_h, [c]_{1,\ell}, \mathsf{pk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{sig}}, \ell), \sigma') = \mathtt{true}.$$

$\mathsf{Sanit}(m, \mathrm{MOD}, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}})$: Return $\perp$ if $\mathsf{Verify}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) \neq \mathtt{true}$. Let $(m^\circ, \sigma_h, [r]^\circ_{1,\ell}, \tau^\circ) \leftarrow \mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, c_h, \mathsf{pk}_{\mathsf{sig}})$, check whether $\mathsf{Verify}_\Sigma(\mathsf{pk}_\Sigma, x_1, \sigma_h) \neq \mathtt{true}$, and return $\perp$ if so. Further, obtain $(h_0^v, \cdot) \leftarrow \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, (0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m^\circ, \tau^\circ, \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r]^\circ_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}); \mathcal{H}(\sigma_h, x_1, \mathsf{pk}_{\mathsf{sig}}))$ and return $\perp$ if $h_0^v \neq h_0$. Otherwise, obtain

$$\forall\, i \in \{1,\ldots,\ell\} \;:\; \mathsf{sk}^i_{\mathsf{ch}} \leftarrow \mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, c_i, \mathsf{pk}_{\mathsf{sig}}) \text{ and return } \perp \textbf{ if } \mathsf{sk}^i_{\mathsf{ch}} = \perp \;\vee\; (m[i]' \in \mathrm{MOD} \;\wedge\; \mathsf{sk}^i_{\mathsf{ch}} = 0).$$

For each block $m[i]' \in \mathrm{MOD}$, let $r_i' \leftarrow \mathsf{CHAdapt}(\mathsf{sk}^i_{\mathsf{ch}}, (i, m[i], \mathsf{pk}_{\mathsf{sig}}), (i, m[i]', \mathsf{pk}_{\mathsf{sig}}), r_i, h_i)$. If any $r_i' = \perp$, return $\perp$. For each block $m[i]' \notin \mathrm{MOD}$, let $r_i' \leftarrow r_i$. Let $m' \leftarrow \mathrm{MOD}(m)$. Draw $\tau' \leftarrow \{0,1\}^{2\lambda}$. Let

$$r_0' \leftarrow \mathsf{CHAdapt}(\mathsf{sk}_{\mathsf{ch}}, (0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}),$$
$$(0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m', \tau', \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r']_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}), r_0, h_0).$$

Finally, return $(m', (\sigma', x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, [r']_{0,\ell}, \tau', c_h, [c]_{1,\ell}, [h]_{0,\ell}))$.

$\mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \mathsf{pk}_{\mathsf{san}})$: If any of the following checks holds return $\perp$:

$$\mathsf{Verify}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) = \mathtt{false} \;\vee\; \exists\, i \in \{1,\ldots,\ell\} \;:\; \mathsf{Verify}(m_i, \sigma_i, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) = \mathtt{false}$$

Otherwise, go through the list of (m_i, σ_i) and find a (non-trivial) colliding tuple of the chameleon-hash with (m, σ), i.e., where it holds that

$$h_0 = h_0' \;\wedge\; \mathtt{true} = \mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, (0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}), r_0, h_0) \;\wedge$$
$$\mathtt{true} = \mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, (0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m', \tau', \ell, [h']_{1,\ell}, c_h', [c']_{1,\ell}, [r']_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}), r_0', h_0') \;\wedge$$
$$(\tau \neq \tau' \;\vee\; m \neq m').$$

Let this signature/message pair be $(\sigma', m') \in \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}$. Return $\pi = ((\sigma', m'), \mathsf{Eval}_{\mathsf{prf}}(\kappa, x_0), \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, x_1))$, where x_0, and x_1, are contained in (σ', m').

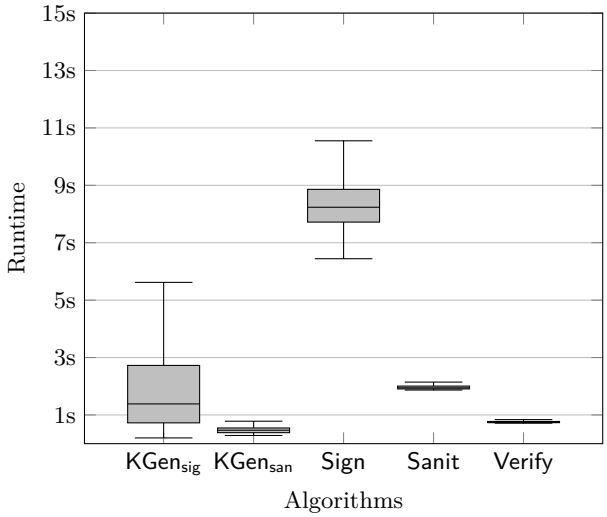
$\mathsf{Judge}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \pi)$: Parse π as $((\sigma', m'), v, \sigma_h)$ with $v \in \{0,1\}^\lambda$, and return Sig on failure. Return Sig if any of the following checks hold

$$\mathtt{false} = \mathsf{Verify}(m', \sigma', \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) \;\vee\; \mathtt{false} = \mathsf{Verify}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}) \;\vee$$
$$\mathsf{Verify}_\Sigma(\mathsf{pk}_\Sigma, x_1, \sigma_h) = \mathtt{false} \;\vee\; \mathsf{Eval}_{\mathsf{prg}}(v) \neq \tau'.$$

With $t' \leftarrow \mathcal{H}(\sigma_h, x_1, \mathsf{pk}_{\mathsf{sig}})$, $(h_t, r_t) \leftarrow \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, (0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m', \tau', \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r']_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}); t')$, return San if we have a non-trivial collision satisfying the following checks, and Sig otherwise:

$$h_0 = h_0' = h_t \;\wedge\; r_t = r_0 \;\wedge\; \mathtt{true} = \mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, (0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, [c]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}), r_0, h_0) \;\wedge$$
$$\mathtt{true} = \mathsf{CHCheck}(\mathsf{pk}'_{\mathsf{ch}}, (0, x_0', x_1', [\mathsf{pk}'_{\mathsf{ch}}]_{1,\ell'}, m', \tau', \ell', [h']_{1,\ell'}, c_h', [c']_{1,\ell'}, \mathsf{pk}_{\mathsf{sig}}), r_0', h_0') \;\wedge\; [c]_{1,\ell} = [c']_{1,\ell'} \;\wedge$$
$$x_0 = x_0' \;\wedge\; x_1 = x_1' \;\wedge\; \ell = \ell' \;\wedge\; [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell} = [\mathsf{pk}'_{\mathsf{ch}}]_{1,\ell'} \;\wedge\; [h]_{0,\ell} = [h']_{0,\ell'}.$$

Construction 2: Secure and transparent SSS



(a) Box-plots of the run-times in ms

	$\mathsf{KGen}_{\mathsf{sig}}$	$\mathsf{KGen}_{\mathsf{san}}$	Sign	Sanit	Verify
Min.:	200	285	6'443	1'868	714
25th PCTL:	725	383	7'718	1'907	731
Median:	1'393	468	8'243	1'946	746
75th PCTL:	2'726	547	8'860	2'006	774
90th PCTL:	4'563	667	9'733	2'161	826
95th PCTL:	5'619	782	10'050	2'317	874
Max.:	8'023	1'018	11'327	3'151	998
Average:	2'017	487	8'367	2'002	764
SD:	1'690	139	914	183	52

(b) Percentiles for our implementation in ms

Fig. 11: Performance Evaluation Results

5 Conclusion

We have strengthened the current invisibility definition of sanitizable signatures. Namely, the adversary is now able to query arbitrary keys to the sanitization oracle. We have shown that prohibiting this may lead to serious problems in real-life scenarios. Moreover, we have presented a simplified construction of a strongly invisible, and transparent, sanitizable signature scheme, which is also *strongly* signer-accountable. That is, we even exclude malleability of the signatures. Our construction is also simpler than the construction given by Camenisch et al. [15], as it does not require chameleon hashes with ephemeral trapdoors, i.e., it only requires standard primitives, which are less costly. Our corresponding evaluation shows that this primitive can be considered practical. Still, it remains an open problem how to construct SSSs which are simultaneously unlinkable, and invisible.

References

1. Ahn, J.H., Boneh, D., Camenisch, J., Hohenberger, S., a. shelat, Waters, B.: Computing on authenticated data. In: TCC. pp. 1–20 (2012)
2. An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: EUROCRYPT 2002. pp. 83–107 (2002)
3. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable signatures. In: ESORICS. pp. 159–177 (2005)
4. Ateniese, G., Magri, B., Venturi, D., Andrade, E.R.: Redactable blockchain - or - rewriting history in bitcoin and friends. IACR Cryptology ePrint Archive 2016, 757 (2016)
5. Ateniese, G., de Medeiros, B.: On the key exposure problem in chameleon hashes. In: SCN. pp. 165–179 (2004)
6. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. J. Cryptology 16(3), 185–215 (2003)
7. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: CCS. pp. 62–73 (1993)
8. Bilzhause, A., Huber, M., Pöhls, H.C., Samelin, K.: Cryptographically Enforced Four-Eyes Principle. In: ARES. pp. 760–767 (2016)
9. Brzuska, C., Busch, H., Dagdelen, Ö., Fischlin, M., Franz, M., Katzenbeisser, S., Manulis, M., Onete, C., Peter, A., Poettering, B., Schröder, D.: Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In: ACNS. pp. 87–104 (2010)
10. Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of Sanitizable Signatures Revisited. In: PKC. pp. 317–336 (2009)
11. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Sanitizable signatures: How to partially delegate control for authenticated data. In: BIOSIG. pp. 117–128 (2009)
12. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Unlinkability of Sanitizable Signatures. In: PKC. pp. 444–461 (2010)

13. Brzuska, C., Pöhls, H.C., Samelin, K.: Non-Interactive Public Accountability for Sanitizable Signatures. In: EuroPKI. pp. 178–193 (2012)
14. Brzuska, C., Pöhls, H.C., Samelin, K.: Efficient and Perfectly Unlinkable Sanitizable Signatures without Group Signatures. In: EuroPKI. pp. 12–30 (2013)
15. Camenisch, J., Derler, D., Krenn, S., Pöhls, H.C., Samelin, K., Slamanig, D.: Chameleon-hashes with ephemeral trapdoors and applications to invisible sanitizable signatures. IACR Cryptology ePrint Archive 2017, 11 (2017)
16. Camenisch, J., Lehmann, A., Neven, G., Samelin, K.: Virtual smart cards: How to sign with a password and a server. In: SCN. pp. 353–371 (2016)
17. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: CRYPTO. pp. 126–144 (2003)
18. Canard, S., Jambert, A.: On extended sanitizable signature schemes. In: CT-RSA. pp. 179–194 (2010)
19. Canard, S., Jambert, A., Lescuyer, R.: Sanitizable signatures with several signers and sanitizers. In: AFRICACRYPT. pp. 35–52 (2012)
20. Canard, S., Laguillaumie, F., Milhau, M.: Trapdoor sanitizable signatures and their application to content protection. In: ACNS. pp. 258–276 (2008)
21. Canard, S., Lescuyer, R.: Protecting privacy by sanitizing personal data: a new approach to anonymous credentials. In: ASIACCS. pp. 381–392 (2013)
22. Damgård, I., Haagh, H., Orlandi, C.: Access control encryption: Enforcing information flow with cryptography. In: TCC-B. pp. 547–576 (2016)
23. Demirel, D., Derler, D., Hanser, C., Pöhls, H.C., Slamanig, D., Traverso, G.: PRISMACLOUD D4.4: Overview of Functional and Malleable Signature Schemes. Tech. rep., H2020 Prismacloud, www.prismacloud.eu (2015)
24. Derler, D., Hanser, C., Slamanig, D.: Blank digital signatures: Optimization and practical experiences. In: Privacy and Identity Management for the Future Internet in the Age of Globalisation, vol. 457, pp. 201–215. Springer (2014)
25. Derler, D., Pöhls, H.C., Samelin, K., Slamanig, D.: A general framework for redactable signatures and new constructions. In: ICISC. pp. 3–19 (2015)
26. Derler, D., Slamanig, D.: Rethinking privacy for extended sanitizable signatures and a black-box construction of strongly private schemes. In: ProvSec. pp. 455–474 (2015)
27. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: PKC. pp. 416–431 (2005)
28. Ejgenberg, Y., Farbstein, M., Levy, M., Lindell, Y.: SCAPI: the secure computation application programming interface. IACR Cryptology ePrint Archive 2012, 629 (2012)
29. Fehr, V., Fischlin, M.: Sanitizable signcryption: Sanitization over encrypted data (full version). IACR Cryptology ePrint Archive, Report 2015/765 (2015)
30. Fleischhacker, N., Krupp, J., Malavolta, G., Schneider, J., Schröder, D., Simkin, M.: Efficient unlinkable sanitizable signatures from signatures with rerandomizable keys. In: PKC-1. pp. 301–330 (2016)
31. Frädrich, C., Pöhls, H.C., Popp, W., Rakotondravony, N., Samelin, K.: Integrity and authenticity protection with selective disclosure control in the cloud & iot. In: ICICS. pp. 197–213 (2016)
32. Ghosh, E., Goodrich, M.T., Ohrimenko, O., Tamassia, R.: Fully-dynamic verifiable zero-knowledge order queries for network data. ePrint 2015, 283 (2015)
33. Ghosh, E., Ohrimenko, O., Tamassia, R.: Zero-knowledge authenticated order queries and order statistics on a list. In: ACNS. vol. 2015, pp. 149–171 (2015)
34. Gong, J., Qian, H., Zhou, Y.: Fully-secure and practical sanitizable signatures. In: Inscrypt. vol. 6584, pp. 300–317 (2011)
35. Hanser, C., Slamanig, D.: Blank digital signatures. In: ASIACCS. pp. 95 – 106 (2013)
36. Höhne, F., Pöhls, H.C., Samelin, K.: Rechtsfolgen editierbarer signaturen. Datenschutz und Datensicherheit 36(7), 485–491 (2012)
37. Johnson, R., Molnar, D., Song, D., D.Wagner: Homomorphic signature schemes. In: CT-RSA. pp. 244–262. Springer (Feb 2002)
38. Kakvi, S.A., Kiltz, E.: Optimal security proofs for full domain hash, revisited. In: EUROCRYPT. pp. 537–553 (2012)
39. Klonowski, M., Lauks, A.: Extended Sanitizable Signatures. In: ICISC. pp. 343–355 (2006)
40. Krenn, S., Samelin, K., Sommer, D.: Stronger security for sanitizable signatures. In: DPM. pp. 100–117 (2015)
41. Lai, R.W.F., Zhang, T., Chow, S.S.M., Schröder, D.: Efficient sanitizable signatures without random oracles. In: ESORICS-1. pp. 363–380 (2016)
42. Lysyanskaya, A.: Unique signatures and verifiable random functions from the DH-DDH separation. In: CRYPTO. pp. 597–612 (2002)
43. de Meer, H., Pöhls, H.C., Posegga, J., Samelin, K.: Scope of security properties of sanitizable signatures revisited. In: ARES. pp. 188–197 (2013)
44. de Meer, H., Pöhls, H.C., Posegga, J., Samelin, K.: On the relation between redactable and sanitizable signature schemes. In: ESSoS. pp. 113–130 (2014)
45. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: FOCS. pp. 120–130 (1999)
46. Miyazaki, K., Hanaoka, G., Imai, H.: Invisibly sanitizable digital signature scheme. IEICE Transactions 91-A(1), 392–402 (2008)

Experiment $\mathsf{Uniqueness}^{\Sigma}_{\mathcal{A}}(\lambda)$
$(\mathsf{pk}^*, m^*, \sigma^*, \sigma'^*) \leftarrow \mathcal{A}(1^\lambda)$
return 1, if $\mathsf{Verify}_\Sigma(\mathsf{pk}^*, m^*, \sigma^*) = \mathtt{true} \ \wedge \ \mathsf{Verify}_\Sigma(\mathsf{pk}^*, m^*, \sigma'^*) = \mathtt{true} \ \wedge \ \sigma^* \neq \sigma'^*$
return 0

Fig. 12: Σ Uniqueness

47. Pöhls, H.C., Peters, S., Samelin, K., Posegga, J., de Meer, H.: Malleable signatures for resource constrained platforms. In: WISTP. pp. 18–33 (2013)
48. Pöhls, H.C., Samelin, K.: Accountable redactable signatures. In: ARES. pp. 60–69 (2015)
49. Pöhls, H.C., Samelin, K., Posegga, J.: Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In: ACNS. LNCS, vol. 6715, pp. 166–182. Springer (2011)
50. Samelin, K., Pöhls, H.C., Bilzhause, A., Posegga, J., de Meer, H.: Redactable signatures for independent removal of structure and content. In: ISPEC. LNCS, vol. 7232, pp. 17–33. Springer (2012)
51. Steinfeld, R., Bull, L., Zheng, Y.: Content extraction signatures. In: Kim, K. (ed.) ICISC. LNCS, vol. 2288, pp. 285–304. Springer (2001)
52. Yum, D.H., Seo, J.W., Lee, P.J.: Trapdoor sanitizable signatures made easy. In: ACNS. pp. 53–68 (2010)

A Security Definitions Building Blocks

This appendix presents the security and correctness definitions required for our construction.

A.1 Digital Signatures Σ

Correctness. For a signature scheme Σ we require the correctness properties to hold. In particular, we require that for all $\lambda \in \mathbb{N}$, for all $(\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma) \leftarrow \mathsf{KGen}_\Sigma(1^\lambda)$, for all $m \in \mathcal{M}$ we have $\mathsf{Verify}_\Sigma(\mathsf{pk}_\Sigma, m, \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, m)) = \mathtt{true}$. This definition captures perfect correctness.

Strong Unforgeability. Now, we define strong unforgeability of digital signature schemes, as given by An et al. [2]. In a nutshell, we require that an adversary $\mathcal{A}$ cannot (except with negligible probability) come up with *any* new valid signature σ^* for a message m^*. Moreover, the adversary $\mathcal{A}$ can adaptively query for new signatures.

Definition 21 (Strong Unforgeability). *A signature scheme Σ is strongly unforgeable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{seUNF\text{-}CMA}^{\Sigma}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 13.*

Uniqueness. Now, we define a computational variant of uniqueness of digital signature schemes [42], derived from Dodis et al., and Micali et al. [27, 45]. In a nutshell, we require that an adversary $\mathcal{A}$ cannot (except with negligible probability) come up with two distinct signatures for some adversarially chosen message, even it can choose the public key itself.

Definition 22 (Uniqueness). *A signature scheme Σ is unique, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Uniqueness}^{\Sigma}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 12.*

A.2 Labeled Public-Key Encryption Schemes Π

Correctness. For a public-key encryption scheme Π we require the correctness properties to hold. In particular, we require that for all $\lambda \in \mathbb{N}$, for all $(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) \leftarrow \mathsf{KGen}_\Pi(1^\lambda)$, for all $m \in \mathcal{M}$, for all $\vartheta \in \{0,1\}^*$, we have $\mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, m, \vartheta), \vartheta) = m$. This definition captures perfect correctness.

Experiment $\mathsf{seUNF\text{-}CMA}^{\Sigma}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma) \leftarrow \mathsf{KGen}_\Sigma(1^\lambda)$
$\mathcal{Q} \leftarrow \emptyset$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}'_\Sigma(\mathsf{sk}_\Sigma, \cdot)}(\mathsf{pk}_\Sigma)$
where oracle Sign'_Σ on input m:
let $\sigma \leftarrow \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, m)$
set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(m, \sigma)\}$
return σ
return 1, if $\mathsf{Verify}_\Sigma(\mathsf{pk}_\Sigma, m^*, \sigma^*) = \mathtt{true} \ \wedge \ (m^*, \sigma^*) \notin \mathcal{Q}$
return 0

Fig. 13: Σ Strong Unforgeability

Experiment $\mathsf{IND\text{-}CCA2}^{\Pi}_{\mathcal{A}}(\lambda)$
$(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) \leftarrow \mathsf{KGen}_\Pi(1^\lambda)$
$b \leftarrow \{0,1\}$
$(m_0, m_1, \vartheta, \mathtt{state}_\mathcal{A}) \leftarrow \mathcal{A}^{\mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, \cdot, \cdot)}(\mathsf{pk}_\Pi)$
if $m_0 \notin \mathcal{M} \ \vee \ m_1 \notin \mathcal{M}$, let $c \leftarrow \perp$
else, let $c \leftarrow \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, m_b, \vartheta)$
$a \leftarrow \mathcal{A}^{\mathsf{Dec}'_\Pi(\mathsf{sk}_\Pi, \cdot, \cdot)}(c, \mathtt{state}_\mathcal{A})$
where oracle $\mathsf{Dec}'_\Pi(\mathsf{sk}_\Pi, \cdot, \cdot)$ behaves as Dec_Π,
but returns $\perp$ if queried with (c, ϑ).
return 1, if $a = b$
return 0

Fig. 14: Π Labeled IND-CCA2-Security

Experiment $\mathsf{Pseudo\text{-}Randomness}^{\mathsf{PRF}}_{\mathcal{A}}(\lambda)$
$\kappa \leftarrow \mathsf{KGen}_{\mathsf{prf}}(1^\lambda)$
$b \leftarrow \{0,1\}$
$f \leftarrow F_\lambda$
$a \leftarrow \mathcal{A}^{\mathsf{Eval}'_{\mathsf{prf}}(\kappa, \cdot)}(1^\lambda)$
where oracle $\mathsf{Eval}'_{\mathsf{prf}}$ on input κ, x:
return $\perp$, if $x \notin \{0,1\}^\lambda$
if $b = 0$, return $\mathsf{Eval}_{\mathsf{prf}}(\kappa, x)$
return $f(x)$
return 1, if $a = b$
return 0

Fig. 15: PRF Pseudo-Randomness

Experiment $\mathsf{Pseudo\text{-}Randomness}^{\mathsf{PRG}}_{\mathcal{A}}(\lambda)$
$b \leftarrow \{0,1\}$
if $b = 0$, let $v \leftarrow \{0,1\}^{2\lambda}$
else, let $x \leftarrow \{0,1\}^\lambda$, and $v \leftarrow \mathsf{Eval}_{\mathsf{prg}}(x)$
$a \leftarrow \mathcal{A}(v)$
return 1, if $a = b$
return 0

Fig. 16: PRG Pseudo-Randomness

Labeled IND-CCA2-Security. Labeled IND-CCA2-Security requires that an adversary $\mathcal{A}$ cannot decide which message is actually contained in a ciphertext c, while $\mathcal{A}$ receives full adaptive access to the decryption oracle. We also require that the message space $\mathcal{M}$ implicitly define an upper bound on the message length, i.e., $|m|$. In other words, this means that the length is implicitly hidden for all messages in $\mathcal{M}$. From a practical viewpoint, this can be implemented using suitable padding techniques.

Definition 23 (Labeled IND-CCA2 Security). *A labeled encryption scheme Π is IND-CCA2-secure, if for any ppt $\mathcal{A}$ there exists a negligible function ν such that* $\left|\Pr[\mathsf{IND\text{-}CCA2}^{\Pi}_{\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 14.*

A.3 PRF Pseudo-Randomness.

In the definition, let $F_\lambda = \{f : \{0,1\}^\lambda \to \{0,1\}^\lambda\}$ be the set of all functions mapping a value $x \in \{0,1\}^\lambda$ to a value $v \in \{0,1\}^\lambda$.

Definition 24 (Pseudo-Randomness). *A pseudo-random function* PRF *is pseudo-random, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\left|\Pr[\mathsf{Pseudo\text{-}Randomness}^{\mathsf{PRF}}_{\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 15.*

A.4 PRG Pseudo-Randomness.

We require that PRG is actually pseudo-random.

Definition 25 (Pseudo-Randomness). *A pseudo-random number-generator* PRG *is pseudo-random, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\left|\Pr[\mathsf{Pseudo\text{-}Randomness}^{\mathsf{PRG}}_{\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 16.*

B Proof of Theorem 1

Now, we prove the security of Construction 2.

Proof. Correctness follows by inspection. We now prove each property on its own.

Unforgeability. To prove that our scheme is unforgeable, we use a sequence of games:

Game 0: The original unforgeability game.

Game 1: As Game 0, but we abort if the adversary outputs a forgery (m^*, σ^*) with $\sigma^* = (\sigma'^*, x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [r^*]_{0,\ell^*}, \tau^*, c_h^*, [c^*]_{1,\ell^*}, [h^*]_{0,\ell^*})$, where $(\sigma'^*, (x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [h^*]_{0,\ell^*}, c_h^*, [c^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{san}}^*, \mathsf{pk}_{\mathsf{sig}}^*, \ell^*))$ was never obtained from the signing oracle. Let this event be E_1.

Transition - Game 0 → Game 1: Clearly, if the tuple $(\sigma'^*, (x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [h^*]_{0,\ell^*}, c_h^*, [c^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{san}}^*, \mathsf{pk}_{\mathsf{sig}}^*, \ell^*))$ was never obtained by the challenger, this tuple breaks the strong unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a strong unforgeability challenger and embed it as $\mathsf{pk}_{\mathsf{sig}}$. For every required "inner" signature σ' (and σ_h), we use the signing oracle provided by the challenger. Now, whenever E_1 happens, we can output σ'^* together with the message protected by σ'^* as a forgery to the challenger. That is, E_1 happens with exactly the same probability as a forgery. Further, both games proceed identically, unless E_1 happens. Taking everything together yields $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{seunf\text{-}cma}}(\lambda)$.

Game 2: Among others, we now have established that the adversary can no longer win by modifying $\mathsf{pk}_{\mathsf{sig}}$, and $\mathsf{pk}_{\mathsf{san}}$. We proceed as in Game 1, but abort if the adversary outputs a forgery (m^*, σ^*), where message m^* or any of the other values protected by the outer chameleon-hash were never returned by the signer or the sanitizer oracle. Let this event be E_2.

Transition - Game 1 → Game 2: The probability of the abort event E_2 to happen is exactly the probability of the adversary breaking collision freeness for the outer chameleon-hash. Namely, we already established that the adversary cannot tamper with the inner signature, and therefore the hash value h_0^* must be from a previous oracle query. Now, assume that we obtain pk_c from a collision freeness challenger. If E_2 happens, there must be a previous oracle query with associated values $(0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}})$ and r_0 so that h_0^* is a valid hash with respect to some those values and r_0. Further, we also have that $(0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}) \neq (0, x_0', x_1', [\mathsf{pk}_{\mathsf{ch}}']_{1,\ell'}, m', \tau', \ell', [h']_{1,\ell'}, c_h', [c']_{1,\ell'}, [r']_{1,\ell'}, \mathsf{pk}_{\mathsf{sig}})$, and can thus output $((0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}), r_0^*, (0, x_0', x_1', [\mathsf{pk}_{\mathsf{ch}}']_{1,\ell'}, m', \tau', \ell', [h']_{1,\ell'}, c_h', [c']_{1,\ell'}, [r']_{1,\ell'}, \mathsf{pk}_{\mathsf{sig}}), r_0'^*, h_0^*)$ as the collision. Thus, the probability that E_2 happens is exactly the probability of a collision for the chameleon-hash. Both games proceed identically, unless E_2 happens. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{ch\text{-}coll\text{-}res}}(\lambda)$ follows.

Game 3: As Game 2, but we abort if the adversary outputs a forgery where only the randomness r_0 changed, i.e., we have previously generated a signature with respect to r_0 so that $r_0 \neq r_0^*$. Let this be event be E_3.

Transition - Game 2 → Game 3: If the abort event E_3 happens, the adversary breaks uniqueness of the chameleon-hash. In particular we have values $(0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}})$ in the forgery which also correspond to some previous query, but r_0 from the previous query is different from r_0^*. Obtaining $\mathsf{pp}_{\mathsf{ch}}$ from a uniqueness challenger thus shows that E_3 happens with exactly the same probability as the adversary breaks uniqueness of the chameleon hash. Thus, we have that $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{ch\text{-}unique}}(\lambda)$.

Now, the adversary can no longer win the unforgeability game; this game is computationally indistinguishable from the original game, which concludes the proof.

Immutability. We prove immutability using a sequence of games.

Game 0: The immutability game.

Game 1: As Game 0, but we abort if the adversary outputs a forgery (m^*, σ^*) with $\sigma^* = (\sigma'^*, x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [r^*]_{0,\ell^*}, \tau^*, c_h^*, [c^*]_{1,\ell^*}, [h^*]_{0,\ell^*})$ where $(\sigma'^*, (x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [h^*]_{0,\ell^*}, c_h^*, [c^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{san}}^*, \mathsf{pk}_{\mathsf{sig}}^*, \ell^*))$ was never obtained from the sign oracle.

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. Clearly, if $(\sigma'^*, (x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [h^*]_{0,\ell^*}, c_h^*, [c^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{san}}^*, \mathsf{pk}_{\mathsf{sig}}^*, \ell^*)$ was never generated by the challenger, this tuple breaks the strong unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a strong unforgeability challenger and embed it as $\mathsf{pk}_{\mathsf{sig}}$. For every required "inner" signature σ' (and σ_h), we use the signing oracle provided by the challenger. Now, whenever E_1 happens, we can output σ'^* together with the message protected by σ'^* as a forgery to the challenger. That is, E_1 happens with exactly the same probability as a forgery of the underlying signature scheme. Further, both games proceed identically, unless E_1 happens. Taking everything together yields $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{seunf\text{-}cma}}(\lambda)$.

Game 2: As Game 1, but the challenger aborts, if the message m^* is not derivable from any returned signature. Let this event be denoted E_2. Note, we already know that tampering with the signatures is not possible, and thus $\mathsf{pk}_{\mathsf{sig}}$, and $\mathsf{pk}_{\mathsf{san}}$, are fixed. The same is true for deleting or appending blocks, as ℓ is signed in every case.

Transition - Game 1 → Game 2: Now assume that E_2 is non-negligible. We can then construct an adversary $\mathcal{B}$ which breaks the collision-resistance of the underlying chameleon-hash. Let the signature returned be $\sigma^* = (\sigma'^*, x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [r^*]_{0,\ell^*}, \tau^*, c_h^*, [c^*]_{1,\ell^*}, [h^*]_{0,\ell^*})$, while $\mathcal{A}$'s public key is pk^*. Due to prior game hops, we know that $\mathcal{A}$ cannot tamper with the "inner" signatures, and *all* hashes are signed. Thus, there must exists another signature $\sigma''^* = (\sigma'''^*, x_0'^*, x_1'^*, [\mathsf{pk}_{\mathsf{ch}}'^*]_{1,\ell'^*}, [r'^*]_{0,\ell'^*}, \tau'^*, c_h'^*, [c'^*]_{1,\ell'^*}, [h'^*]_{0,\ell'^*})$ returned by the signing oracle. This, however, also implies that there must exists an index $i \in \{1, 2, \ldots, \ell^*\}$ (as $\ell^* = \ell'^*$), for which we have $\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, (i, m^*[i], \mathsf{pk}_{\mathsf{sig}}), r_i^*, h_i^*) = \mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, (i, m'^*[i], \mathsf{pk}_{\mathsf{sig}}), r_i'^*, h_i^*) = \mathtt{true}$, where $m^*[i] \neq m'^*[i]$ by assumption. For the reduction, $\mathcal{B}$ proceeds as follows. Let q_h be the number of "inner hashes" created. Draw an index $i \leftarrow \{1, 2, \ldots, q_h\}$. For a query $i \neq j$, proceed as in the algorithms. If $i = j$, however, $\mathcal{B}$ returns the challenge public key pk_c for the chameleon-hash. Note, the ciphertexts can be honestly generated. $\mathcal{B}$ then receives back control, and queries its CHash oracle with $(i, m[i], \mathsf{pk}_{\mathsf{sig}})$, where i is the current index of block of the message m to be signed. Then, if $((i, m^*[i], \mathsf{pk}_{\mathsf{sig}}), r_i^*, (i, m'^*[i], \mathsf{pk}_{\mathsf{sig}}), r_i'^*, h_i^*)$ is the collision w.r.t. pk_c, it can directly return it. $|\Pr[S_1] - \Pr[S_2]| \leq q_h \nu_{\mathsf{ch\text{-}coll\text{-}res}}(\lambda)$ follows, as $\mathcal{B}$ has to guess where the collision will take place.

As each hop changes the view of the adversary only negligibly, immutability is proven, as the adversary has no other way to break immutability in Game 2.

Privacy. We now prove privacy; we use a sequence of games. Note, the adversary never sees the non-sanitized versions of the signature generated by $\mathsf{LoRSanit}$. Thus, the Proof oracle cannot be queried to receive a proof π for such signatures.

Game 0: The original privacy game.

Game 1: As Game 0, but we abort if the adversary queries a verifying message-signature pair (m^*, σ^*) which was never returned by the signer, or the sanitizer, oracle, to the sanitization, or proof generation, oracle.

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. Clearly, whenever the adversary queries such a new pair, we can output it to break the unforgeability of our scheme, as this tuple is fresh. However, we have already proven that this can only happen with negligible probability. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{sss\text{-}unf}}(\lambda)$ follows.

Game 2: As Game 1, but all c_h encrypt a 0 in the $\mathsf{LoRSanit}$ oracle. For sanitization of those signatures, the values are stored, and then used without decrypting c_h. However, the oracle still enforces the correct ADM.

Transition - Game 1 → Game 2: A standard reduction, using hybrids, shows that this hop is indistinguishable by the IND-CCA2-Security of the encryption scheme used. $|\Pr[S_1] - \Pr[S_2]| \leq q_h \nu_{\mathsf{ind\text{-}cca2}}(\lambda)$ follows, where q_h is the number of generated ciphertexts by $\mathsf{LoRSanit}$. Note, requests for other $\mathsf{pk}_{\mathsf{sig}}$s can simply be decrypted using the decryption oracle provided.

Game 3: As Game 2, but we abort if a x_1 was drawn twice. Let this event be E_3.

Transition - Game 2 → Game 3: As each x_1 is drawn uniformly at randomly, event E_3 only happens with probability $\frac{q_t^2}{2\lambda}$, where q_t is the number of signature generation requests. $|\Pr[S_2] - \Pr[S_3]| \leq \frac{q_t^2}{2^\lambda}$ follows.

Game 4: As Game 3, but we no longer compute the signatures σ_h and choose $t \leftarrow \{0,1\}^\lambda$ when simulating the $\mathsf{LoRSanit}$ oracle.

Transition - Game 3 → Game 4: We can construct an adversary $\mathcal{B}$ which breaks the strong unforgeability of the signature scheme from a distinguisher between Game 3 and Game 4. In the first step, $\mathcal{B}$ takes over control of the random oracle $\mathcal{H}$, and simulates honestly, but records the queries. It then receives pk_c as the challenge key, which is embedded into $\mathsf{pk}_{\mathsf{sig}}$. Every required signature generation is delegated to the signing oracle provided. However, as we have already ruled out collision for the x_1, we must have a query $(x_1, \sigma_h, \mathsf{pk}_{\mathsf{sig}})$ where σ_h is valid, fresh signature on x_1. $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{seunf\text{-}cma}}(\lambda)$ follows.

Game 5: As Game 4, but instead of hashing the blocks $(i, m_b[i], \mathsf{pk}_{\mathsf{sig}})$ for the inner chameleon-hashes using CHash, and then $\mathsf{CHAdapt}$ to $(i, m[i], \mathsf{pk}_{\mathsf{sig}})$, we directly apply CHash to $(i, m[i], \mathsf{pk}_{\mathsf{sig}})$.

Transition - Game 4 → Game 5: Assume that the adversary can distinguish this hop. We can then construct an adversary $\mathcal{B}$ which wins the indistinguishability game. $\mathcal{B}$ receives pk_c as it's own challenge, $\mathcal{B}$ embeds pk_c as one of the $\mathsf{pk}_{\mathsf{ch}}^i$ (where $m[i]$ is admissible). It proceeds honestly with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate that inner hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_4] - \Pr[S_5]| \leq q_h \nu_{\mathsf{ch\text{-}ind}}(\lambda)$ follows, where q_h is the number of admissible inner hashes due to a standard hybrid argument.

Game 6: As Game 5, but instead of adapting $(0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, [c]_{1,\ell}, [r]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}})$ to the new values, we directly use CHash.

Transition - Game 5 → Game 6: Assume that the adversary can distinguish this hop. We can then construct an $\mathcal{B}$ which wins the indistinguishability game. $\mathcal{B}$ receives pk_c as it's own challenge, $\mathcal{B}$ embeds pk_c as $\mathsf{pk}_{\mathsf{ch}}$, and proceeds honestly with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate the outer hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_5] - \Pr[S_6]| \leq \nu_{\mathsf{ch\text{-}ind}}(\lambda)$ follows.

Clearly, we are now independent of the bit b. As each hop changes the view of the adversary only negligibly, privacy is proven.

Transparency. We prove transparency by showing that the distributions of sanitized and fresh signatures are indistinguishable. Note, the adversary is not allowed to query Proof for values generated by $\mathsf{Sanit/Sign}$.

Game 0: The original transparency game with $b = 0$.

Game 1: As Game 0, but we abort if the adversary queries a valid message-signature pair (m^*, σ^*) which was never returned by any of the calls to the sanitization or signature generation oracle. Let us use E_1 to refer to the abort event.

Transition - Game 0 → Game 1: Clearly, whenever the adversary queries such a new pair, we can output it to break the unforgeability of our scheme, as this tuple is fresh. A reduction is straightforward. Thus, we have $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{sss\text{-}unf}}(\lambda)$.

Game 2: As Game 1, but instead of computing $x_0' \leftarrow \mathsf{Eval}_{\mathsf{prf}}(\kappa, x_0)$, we set $x_0' \leftarrow \{0,1\}^\lambda$ within every call to Sign in the $\mathsf{Sanit/Sign}$ oracle.

Transition - Game 1 → Game 2: A distinguisher between these two games straightforwardly yields a distinguisher for the PRF. Thus, we have $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{ind\text{-}prf}}(\lambda)$.

Game 3: As Game 2, but instead of computing $\tau \leftarrow \mathsf{Eval}_{\mathsf{prg}}(x_0')$, we set $\tau \leftarrow \{0,1\}^{2\lambda}$ for every call to Sign within the $\mathsf{Sanit/Sign}$ oracle.

Transition - Game 2 → Game 3: A distinguisher between these two games yields a distinguisher for the PRG using a hybrid argument. Thus, we have $|\Pr[S_2] - \Pr[S_3]| \leq q_s \nu_{\mathsf{ind\text{-}prg}}(\lambda)$, where q_s is the number of calls to the PRG.

Game 4: As Game 3, but we abort if a tag τ was drawn twice. Let this event be E_4.

Transition - Game 3 → Game 4: As the tags τ are drawn uniformly random, event E_4 only happens with probability $\frac{q_t^2}{2^{2\lambda}}$, where q_t is the number of drawn tags. $|\Pr[S_3] - \Pr[S_4]| \leq \frac{q_t^2}{2^{2\lambda}}$ follows.

Game 5: As Game 4, but instead of hash, and then adapting, the inner chameleon-hashes, directly hash $(i, m[i], \mathsf{pk}_{\mathsf{sig}})$.

Transition - Game 4 → Game 5: Assume that the adversary can distinguish this hop. We can then construct an adversary $\mathcal{B}$ which wins the indistinguishability game. $\mathcal{B}$ receives pk_c as it's own challenge, $\mathcal{B}$ embeds pk_c as one of the $\mathsf{pk}_{\mathsf{ch}}^i$ (where $m[i]$ is admissible). It proceeds honestly with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate that inner hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_4] - \Pr[S_5]| \leq q_h \nu_{\mathsf{ch\text{-}ind}}(\lambda)$ follows, where q_h is the number of admissible inner hashes due to a standard hybrid argument, i.e., exchange the public keys one-by-one.

Game 6: As Game 5, but we abort if a x_1 was drawn twice. Let this event be E_6.

Transition - Game 5 → Game 6: As each x_1 is drawn completely randomly, event E_6 only happens with probability $\frac{q_t^2}{2^{\lambda}}$, where q_t is the number of signatures requests. $|\Pr[S_5] - \Pr[S_6]| \leq \frac{q_t^2}{2^{\lambda}}$ follows.

Game 7: As Game 6, but all c_h encrypt a 0 in the $\mathsf{Sanit/Sign}$ oracle. For sanitization of those signatures, the values are stored, and then used without decrypting c_h. However, the oracle still enforces the correct ADM.

Transition - Game 6 → Game 7: A standard reduction, using hybrids, shows that this hop is indistinguishable by the IND-CCA2-Security of the encryption scheme used. $|\Pr[S_6] - \Pr[S_7]| \leq q_h \nu_{\mathsf{ind\text{-}cca2}}(\lambda)$ follows, where q_h is the number of generated ciphertexts by $\mathsf{Sanit/Sign}$. Note, requests for other $\mathsf{pk}_{\mathsf{sig}}$s can simply be decrypted using the decryption oracle provided.

Game 8: As Game 7, but we no longer sign the values x_1 in $\mathsf{Sanit/Sign}$ and obtain $t \leftarrow \{0,1\}^{\lambda}$.

Transition - Game 7 → Game 8: A distinguisher between Game 7 and Game 8, must have made a random-oracle call $(x_1, \sigma_h, \mathsf{pk}_{\Sigma})$ such that $\mathsf{Verify}_{\Sigma}(\mathsf{pk}_{\Sigma}, x_1, \sigma_h) = \mathtt{true}$. Let this event be E_8. We can then construct an adversary $\mathcal{B}$ which breaks the strong unforgeability of the signature scheme with $\Pr[E_8]$. In the first step, $\mathcal{B}$ takes over control of the random oracle $\mathcal{H}$, and simulates honestly, but records the queries. It then receives pk_c as the challenge key, which is embedded into pk_{Σ}. Every required signature generation is delegated to the signing oracle provided. However, as we have already ruled out collision for the x_1 we have a valid forgery for the signature scheme. $|\Pr[S_7] - \Pr[S_8]| \leq \nu_{\mathsf{seunf\text{-}cma}}(\lambda)$ follows.

Game 9: As Game 8, but instead of hashing and then adapting the outer hash, we directly hash the message, i.e., $(0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}})$.

Transition - Game 8 → Game 9: Assume that the adversary can distinguish this hop. We can then construct an $\mathcal{B}$ which wins the indistinguishability game. In particular, the reduction works as follows. $\mathcal{B}$ receives pk_c as it's own challenge, embeds pk_c as $\mathsf{pk}_{\mathsf{ch}}$, and proceeds honestly with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate the outer hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_8] - \Pr[S_9]| \leq \nu_{\mathsf{ind\text{-}ch}}(\lambda)$ follows.

We are in the case $b = 1$, yet do not give out any useful information at all. As each hop only changes the view negligibly, transparency is proven.

Strong Signer-Accountability. We prove that our construction is strongly signer-accountable by a sequence of games.

Game 0: The original strong signer-accountability game.

Game 1: As Game 0, but we abort if the forgery of the signer contains two distinct "inner" signatures, i.e., σ''^* contained in π^* and σ'^* contained in σ^*, on the same message $m^* = (x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [r^*]_{0,\ell^*}, \tau^*, c_h^*, [c^*]_{1,\ell^*}, [h^*]_{0,\ell^*})$ contained in π^*, or the sanitization oracle saw a different "inner" signature (for the same pk^*) as provided in π^*. Let this event be E_1.

Transition - Game 0 → Game 1: Assume that E_1 is non-negligible. We can then construct an adversary $\mathcal{B}$ which breaks the uniqueness of the signature scheme. Namely, we have σ'^* and $\sigma''^* \neq \sigma'^*$ which are valid for the same message $m^* = (x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [r^*]_{0,\ell^*}, \tau^*, c_h^*, [c^*]_{1,\ell^*}, [h^*]_{0,\ell^*})$. Thus, $\mathcal{B}$ can return $(\mathsf{pk}_{\Sigma}^*, m^*, \sigma''^*, \sigma'^*)$ as its own forgery attempt. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{dsig\text{-}uniqueness}}(\lambda)$ follows. pk_{Σ}^* is contained in pk^*.

Game 2: As Game 1, but we abort if the sanitization oracle draws a tag τ' which is in the range of the PRG. Let this event be E_2.

Transition - Game 1 → Game 2: This hop is indistinguishable by a standard statistical argument: at most 2^λ values lie in the range of the PRG. $|\Pr[S_1] - \Pr[S_2]| \leq \frac{q_s 2^\lambda}{2^{2\lambda}} = \frac{q_s}{2^\lambda}$ follows, where q_s is the number of sanitizing requests. Note, this also means, that there exists no valid pre-image x_0.

Game 3: As Game 2, but we now abort, if a tag τ was drawn twice. Let this event be E_3.

Transition - Game 2 → Game 3: As the tags are drawn uniformly from $\{0,1\}^{2\lambda}$, this case only happens with negligible probability. $|\Pr[S_2] - \Pr[S_3]| \leq \frac{q_s^2}{2^{2\lambda}}$ follows, where q_s is the number of sanitization oracle queries.

Game 4: As Game 3, but we now abort, if the adversary was able to find $(\mathsf{pk}^*, \pi^*, m^*, \sigma^*)$ for some $(0, x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell}, m^*, \tau^*, \ell^*, [h^*]_{1,\ell}, c_h^*, [c^*]_{1,\ell}, [r^*]_{1,\ell}, \mathsf{pk}^*)$ in (m^*, σ^*), which was never returned by the sanitization oracle. Let this event be E_4.

Transition - Game 3 → Game 4: In the previous games we have already established that the sanitizer oracle will never return a signature with respect to a tag τ in the range of the PRG. Thus, if event E_4 happens, we know by the conditions checked in Game 2, and Judge, that one of the tags (i.e., τ^π in π^* by construction) was chosen by the adversary, which, in further consequence, implies a collision for CH. Namely, assume that E_3 happens with non-negligible probability. Then we embed the challenge public key pk_c in $\mathsf{pk}_{\mathsf{ch}}$, and use the provided adaption oracle to simulate the sanitizing oracle. If E_4 happens we can output $((0, x_0, x_1, [\mathsf{pk}_{\mathsf{ch}}]_{1,\ell}, m, \tau, \ell, [h]_{1,\ell}, c_h, [c]_{1,\ell}, [r]_{1,\ell}, \mathsf{pk}^*), r_0, (0, x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell}, m^*, \tau^*, \ell^*, [h^*]_{1,\ell}, c_h^*, [c^*]_{1,\ell}, [r^*]_{1,\ell}, \mathsf{pk}^*), r_0', h_0)$, as a valid collision. These values can simply be compiled using π^*, m^*, and σ^*. Note, this also means that the adversary cannot tamper with the "inner" hashes, while each message returned by the sanitization oracle is new, as we excluded tag-collisions. $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{ch\text{-}coll\text{-}res}}(\lambda)$ follows.

Game 5: As Game 4, but abort, if the adversary outputs a forgery, where only the hash h_0 is modified. Let this event be E_5.

Transition - Game 4 → Game 5: As we have already ruled out collisions not provided by the collision-finding oracle, there must be two different hashes h_0 for the outer hash. However, since the CHash algorithm is deterministic with respect to a fixed random tape t (which is uniquely determined by π^*). This implies that there exist two signatures $\sigma_h^* \neq \sigma_h'^*$ where $(\sigma_h^*, x_1^*, \mathsf{pk}^*)$, and $(\sigma_h'^*, x_1^*, \mathsf{pk}^*)$, must yield different hashes. Both are valid signatures for the same x_1^* under pk^*, as otherwise Judge does not output San. Note, altering the other values have already been excluded in the prior hop, as they are hashed and signed. Both signatures can simply be extracted from π^*, and the transcript to the sanitization oracle. This is possible, as the sanitization oracle only proceeds if the randomness is calculated correctly, while Judge enforces that all values are equal, and we already established that collisions can only be produced by the sanitization oracle. These signatures can directly be used to break the uniqueness of the signature scheme w.r.t. to pk_Σ^*, contained in pk^*, and x_1^*. $|\Pr[S_4] - \Pr[S_5]| \leq \nu_{\mathsf{dsig\text{-}unique}}(\lambda)$ follows.

Game 6: As Game 5, but we abort if the adversary outputs a forgery where only the randomness r_0 changed, i.e., we have previously generated a signature with respect to r_0 so that $r_0 \neq r_0^*$. Let this event be E_6.

Transition - Game 5 → Game 6: If the abort event E_6 happens, the adversary breaks uniqueness of the chameleon-hash. In particular, we have values $(0, x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, m^*, \tau^*, \ell^*, [h^*]_{1,\ell^*}, c_h^*, [c^*]_{1,\ell^*}, [r^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{sig}})$ in the forgery which also correspond to some previous query, but r_0 from the previous query is different from r_0^*. Obtaining $\mathsf{pp}_{\mathsf{ch}}$ from a uniqueness challenger thus shows that E_5 happens with exactly the same probability as the adversary breaks uniqueness of the chameleon hash along with $\mathsf{pk}_{\mathsf{ch}}^*$ contained in pk^*. Thus, we have that $|\Pr[S_5] - \Pr[S_6]| \leq \nu_{\mathsf{ch\text{-}unique}}(\lambda)$.

In the last game the adversary can no longer win, and each hop only changes the view negligibly. This concludes the proof.

Sanitizer-Accountability. We prove that our construction is sanitizer-accountable by a sequence of games.

Game 0: The original sanitizer-accountability definition.

Game 1: As Game 0, but we abort if the adversary outputs a forgery $(m^*, \sigma^*, \mathsf{pk}^*)$ with $\sigma^* = (\sigma'^*, x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [r^*]_{0,\ell^*}, \tau^*, c_h^*, [c^*]_{1,\ell^*}, [h^*]_{0,\ell^*})$, where $(\sigma'^*, (x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [h^*]_{0,\ell^*}, c_h^*, [c^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{san}}^*, \mathsf{pk}_{\mathsf{sig}}^*, \ell^*))$ was never obtained from the signing oracle.

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. Clearly, if $(\sigma'^*, (x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, [h^*]_{0,\ell^*}, c_h^*, [c^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{san}}^*, \mathsf{pk}_{\mathsf{sig}}^*, \ell^*))$ was never obtained by the challenger, this tuple breaks the strong unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a strong unforgeability challenger and embed it as $\mathsf{pk}_{\mathsf{sig}}$. For every required "inner" signature σ' (and σ_h), we use the signing oracle provided by the challenger. Now, whenever E_1 happens, we can output σ'^* together with the message protected by σ'^* as a forgery to the challenger. That is, E_1 happens with exactly the same probability as a forgery. Further, both games proceed identically, unless E_1 happens. Taking everything together yields $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{seunf\text{-}cma}}(\lambda)$.

Game 2: As Game 1, but we abort if the adversary outputs a forgery where only the randomness r_0 changed, i.e., we have previously generated a signature with respect to r_0 so that $r_0 \neq r_0^*$. Let this event be E_2.

Transition - Game 1 → Game 2: If the abort event E_2 happens, the adversary breaks uniqueness of the chameleon-hash. In particular, we have values $(0, x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, m^*, \tau^*, \ell^*, [h^*]_{1,\ell^*}, c_h^*, [c^*]_{1,\ell^*}, [r^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{sig}})$ in the forgery which also correspond to some previous query, but r_0 from the previous query is different from r_0^*. Obtaining $\mathsf{pp}_{\mathsf{ch}}$ from a uniqueness challenger thus shows that E_2 happens with exactly the same probability as the adversary breaks uniqueness of the chameleon hash along with $\mathsf{pk}_{\mathsf{ch}}^*$ contained in pk^*. Thus, we have that $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{ch\text{-}unique}}(\lambda)$.

In Game 2 the forgery is different from any query/answer tuple obtained using Sign. Due to the previous hops, the only remaining possibility is a collision for $h_0^* = h_0'^*$, i.e., we have $\mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, (0, x_0^*, x_1^*, [\mathsf{pk}_{\mathsf{ch}}^*]_{1,\ell^*}, m^*, \tau^*, \ell^*, [h^*]_{1,\ell^*}, c_h^*, [c^*]_{1,\ell^*}, [r^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{sig}}), r_0^*, h_0^*) = \mathsf{CHCheck}(\mathsf{pk}_{\mathsf{ch}}, (0, x_0'^*, x_1'^*, [\mathsf{pk}_{\mathsf{ch}}'^*]_{1,\ell'^*}, m'^*, \tau'^*, \ell'^*, [h'^*]_{1,\ell'^*}, c_h'^*, [c'^*]_{1,\ell'^*}, [r'^*]_{1,\ell'^*}, \mathsf{pk}_{\mathsf{sig}}), r_0'^*, h_0'^*) = \mathtt{true}$. In this case, the Judge algorithm returns San and $\Pr[S_2] = 0$ which concludes the proof.

Strong Invisibility. We prove that our construction is strongly invisible by a sequence of games.

Game 0: The original invisibility game, i.e., the challenger runs the experiment as defined.

Game 1: As Game 0, but we abort if the adversary queries a valid message-signature pair (m^*, σ^*) which was never returned by the signer or the sanitizer oracle to the sanitization or proof generation oracle.

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. Clearly, whenever the adversary outputs such a new pair, we can output it to break unforgeability of our scheme, as this tuple is fresh. However, we have already proven that this can only happen with negligible probability. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{sss\text{-}unf}}(\lambda)$ follows.

Game 2: As Game 1, but we internally keep all $\mathsf{sk}_{\mathsf{ch}}^i$.

Transition - Game 1 → Game 2: This is only a conceptual change. $|\Pr[S_1] - \Pr[S_2]| = 0$ follows.

Game 3: As Game 2, but we encrypt only zeroes instead of the real $\mathsf{sk}_{\mathsf{ch}}^i$ in LoRADM independent of whether block are admissible or not, if $\mathsf{pk}_{\mathsf{san}} = \mathsf{pk}_{\mathsf{san}}'$. Note, the challenger still knows all $\mathsf{sk}_{\mathsf{ch}}^i$, and can thus still sanitize correctly.

Transition - Game 2 → Game 3: A standard reduction, using hybrids, shows that this hop is indistinguishable by the IND-CCA2-Security of the encryption scheme used. $|\Pr[S_2] - \Pr[S_3]| \leq q_h \nu_{\mathsf{ind\text{-}cca2}}(\lambda)$ follows, where q_h is the number of generated ciphertexts by LoRADM. Note, requests for other $\mathsf{pk}_{\mathsf{sig}}$s can simply be decrypted using the decryption oracle provided. This is possible, as $\mathsf{pk}_{\mathsf{sig}}$ is part of the label and signed. Thus, the label must be different for any other $\mathsf{pk}_{\mathsf{sig}}$, even if the adversary "re-uses" ciphertexts. This allows to simulate correctly, as the views are equal.

At this point, the distribution is independent of the LoRADM oracle, i.e., completely independent of the bit b. As each hop only changes the view of the adversary negligibly, our construction is thus strongly invisible. □