This is the full and revised version of a paper which appears in Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings. LNCS 10831. pages 11–32 © Springer, 2018.
The final authenticated version is available online at https://doi.org/10.1007/978-3-319-89339-6_2

Chameleon-Hashes with Dual Long-Term Trapdoors and Their Applications[‡]

Stephan Krenn[1], Henrich C. Pöhls[2], Kai Samelin[3‖], and Daniel Slamanig[1]

[1] AIT Austrian Institute of Technology, Vienna, Austria
{stephan.krenn,daniel.slamanig}@ait.ac.at
[2] ISL & Chair of IT-Security, University of Passau, Passau, Germany
hp@sec.uni-passau.de
[3] TÜV Rheinland i-sec GmbH, Hallbergmoos, Germany
kaispapers@gmail.com

Abstract. A chameleon-hash behaves likes a standard collision-resistant hash function for outsiders. If, however, a trapdoor is known, arbitrary collisions can be found. Chameleon-hashes with ephemeral trapdoors (CHET; Camenisch et al., PKC '17) allow prohibiting that the holder of the long-term trapdoor can find collisions by introducing a second, ephemeral, trapdoor. However, this ephemeral trapdoor is required to be chosen freshly for *each* hash.

We extend these ideas and introduce the notion of chameleon-hashes with *dual long-term* trapdoors (CHDLTT). Here, the second trapdoor is not chosen freshly for each new hash; Rather, the hashing party can decide if it wants to generate a fresh second trapdoor or use an existing one. This primitive generalizes CHETs, extends their applicability and enables some appealing new use-cases, including *three-party* sanitizable signatures, group-level selectively revocable signatures and break-the-glass signatures. We present two provably secure constructions and an implementation which demonstrates that this extended primitive is efficient enough for use in practice.

1 Introduction

Standard chameleon-hashes have proven to be useful in very different areas such as on/offline signatures [28, 33, 57], (tightly) secure signatures [14, 44, 51], but also sanitizable signature [2, 17, 22, 41] and identity-based encryption [59]. They are also useful in context of trapdoor-commitments, direct anonymous attestation, Σ-protocols and distributed hashing [1, 10, 16]. Recently, they have been extended to have ephemeral trapdoors, which allow one to find collisions if, and only if, two trapdoors at the same time are known [22]. One of those trapdoors is long-term, while the second one is chosen freshly for each new hash.

The first application of these so called CHETs were invisible sanitizable signatures [7, 22]. In this primitive, a semi-trusted third party, named the sanitizer having its own key pair, can modify signer-chosen admissible parts of a signed message to arbitrary bit-strings [2, 17]. The derived signatures still verify, while an outsider cannot decide which parts are actually modifiable. However, the current formalization of those chameleon-hashes inherently requires that the ephemeral trapdoor is chosen freshly for *each* new hash. Thus, for each generated hash, the hashing party can decide if, and when, the holder of the long-term trapdoor can find collisions. This requirement, however, is quite restricting and once it is lifted, several new interesting use-cases are possible.

[‡] The project leading to this work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644962 PRISMACLOUD and No 321310 PERCY.

[‖] Part of this work was done while the third author was also at IBM Research – Zurich and TU Darmstadt.

Motivation and Contribution. We introduce chameleon-hashes with dual long-term trapdoors (CHDLTT). In this primitive, the second trapdoor no longer needs to be freshly generated on every hash computation, but can be re-used in several hash computations. This has several advantages over CHETs as defined by Camenisch et al. [22]. For example, our definitions allow that the trapdoors can be generated in advance (and thus can, e.g., be registered at a PKI before usage), re-used multiple times (which adds flexibility and saves computational costs) and are inherently independent of any other hashing keys created. In other words, releasing the second trapdoor allows the holder of the long-term secret to find collisions for *all* hashes created using the corresponding second public hashing key that corresponds to the trapdoor. Thus, our new primitive strictly generalizes CHETs: using a fresh trapdoor for each generated hash within our extended definition of CHDLTT resembles the behavior of a CHET.

We introduce a suitable framework, along with corresponding security definitions and two provably secure constructions. The first construction is based on standard collision-resistant chameleon-hashes, while the second one is based on the one-more RSA-Assumption. To demonstrate the usefulness of our new primitive, we show how one can construct *three*-party sanitizable signatures. In this new primitive, a *signer*-designated third party can decide if, and when, the sanitizer is able to sanitize the admissible parts of messages. This new primitive enables several new use-cases for sanitizable signatures scheme. For example, consider the following scenario, where "standard" sanitizable signatures [2] are not sufficient. Assume a supply chain, where the delivery person needs to claim that it delivered the goods as instructed by the sender, while also the recipient needs to vouch for the correct delivery at time of successful reception. In standard sanitizable signature schemes, either a fresh signature is generated at delivery, which is approved by the recipient, or vice versa. However, in this case, the sender has no control what is actually signed. Our extended primitives allows to tackle this problem: the sender can sign what has been commissioned, while the delivery person can approve that it has successfully delivered the goods, if, and only if, the recipient allows the delivery man to do this, i.e., by releasing an additional trapdoor. To some extend, this resembles "four-eyes"-signatures [12], but in a more sophisticated manner, as in our case also damaged goods can be reported *within the same report*, e.g., using special blocks of the message.

Additional application scenarios include group-level selectively revocable signatures and break-the-glass signatures. Revocable signatures have already been mentioned by Camenisch et al. [22] as a potential additional application scenario, while break-the-glass signatures are a new concept. Moreover, our evaluation shows that the primitive is practically efficient. Summarized, our work leads to some new interesting applications of chameleon-hashes, which may give rise to new use-cases which have not been considered yet.

Related Work and State-of-the-Art. Chameleon-hashes were introduced by Krawczyk and Rabin [47], based on the work done by Brassard et al. [16]. Later, they have been ported to the identity-based setting, i.e., ID-based chameleon-hash functions, where the holder of some master secret key can extract new secret keys for new identities [4, 6, 56, 58]. However, most of the schemes presented suffer from the key-exposure problem [5, 27, 47]. Key-exposure means that seeing a single collision in the hash allows to find further collisions by extracting the corresponding trapdoor, i.e., the secret key.[4] This problem was addressed by the introduction of "key-exposure free" chameleon-hashes [5, 26, 27, 38, 39, 56], which prohibit extracting the secret key if a collision was seen. This also includes combinations of both techniques [29]. Such schemes allow for re-using secret key-material. However, the definition of collision-resistance is defined w.r.t. some additional label L.[5] This label L is used to define a "collision-domain", i.e., such hashes do *not* prohibit that once a collision is made public for a label L, an adversary cannot produce additional collisions w.r.t. that label L. We stress that our framework does not require such a label, i.e., it is collision-resistant in the "usual" sense. Brzuska et al. also proposed a formal framework for tag-based chameleon-hashes secure under random-tagging attacks, i.e., they

[4] In the case of identity-based chameleon-hashes w.r.t. to some identity.
[5] Also referred to as nonce or tag.

add an additional *random* tag to the input of the hashing algorithm [17]. We stress that all of the mentioned approaches are orthogonal to our primitive, as in our case two keys at *the same time* are required. Camenisch et al. presented a new type of chameleon-hash, where the hashing party can prohibit that the holder of the long-term secret can find collisions by adding a second, i.e., ephemeral, trapdoor [22], which is chosen freshly for each new hash. Their work can be seen as the starting point for our work. Additional related work is discussed when presenting the applications of our new primitive.

2 Preliminaries

Let us give our notation and assumptions first. Additional standard formal security definitions are given in App. A.

Notation. $\lambda \in \mathbb{N}$ denotes our security parameter. 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. An algorithm is efficient if it runs in probabilistic polynomial time (ppt) in the length of its input. All algorithms are ppt, if not explicitly mentioned otherwise. Most algorithms may return a special error symbol $\perp \notin \{0,1\}^*$, denoting an exception. Returning output ends an algorithm. If S is a set, we write $a \leftarrow S$ to denote that a is chosen uniformly at random from S. For a list we require that there is an 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 $\nu(n) \leq n^{-k}$, $\forall n > n_0$. Moreover, we require that one can derive a public key from the private key. This is not explicitly checked. This can be achieved by appending the randomness used to generate the key pair to the secret key.

The One-More-RSA-Assumption [9]. Let $(n, e, d, p, q) \leftarrow \mathsf{RSA}(1^\lambda)$ be an RSA-key generator returning an RSA modulus $n = pq$, where p and q are random distinct primes, $e > 1$ an integer co-prime to $\varphi(n)$, and $d \equiv e^{-1} \bmod \varphi(n)$. The one-more-RSA-assumption associated to RSA is provided an inversion oracle $\mathcal{I}$, which inverts any element $x \in \mathbb{Z}_n^*$ w.r.t. e, and a challenge oracle $\mathcal{C}$, which at each call returns a random element $y_i \in \mathbb{Z}_n^*$. An adversary wins if, given n and e, it is able to invert more elements received by $\mathcal{C}$ than is makes calls to $\mathcal{I}$. The corresponding assumption states that for every ppt adversary $\mathcal{A}$ there exists a negligible function ν such that:

$$\Pr[(n,p,q,e,d) \leftarrow \mathsf{RSA}(1^\lambda), X \leftarrow \mathcal{A}(n,e)^{\mathcal{C}(n),\mathcal{I}(d,n,\cdot)} : \\ \text{more values returned by } \mathcal{C} \text{ are inverted than queries to } \mathcal{I}] \leq \nu(\lambda)$$

Here, X is the set of inverted challenges.

We require that e is larger than any possible n w.r.t. λ and that it is prime. Re-stating the assumption with this condition is straightforward. In this case, it is also required that e is drawn independently from p, q, or n (and d is then calculated from e, and not vice versa). This can, e.g., be achieved by demanding that e is drawn uniformly from $[n'+1, \ldots, 2n'] \cap \{p \mid p \text{ is prime}\}$, where n' is the largest RSA modulus possible w.r.t. to λ. The details are left to the concrete instantiation of RSA.

Chameleon-Hashes. The given framework is the one by Camenisch et al. [22].

Definition 1. *A chameleon-hash* CH *consists of five algorithms* $(\mathsf{PGen}, \mathsf{KGen}, \mathsf{Hash}, \mathsf{Ver}, \mathsf{Adap})$*, such that:*

PGen. *It outputs the public parameters of the scheme:*

$$\mathsf{pp_{ch}} \leftarrow \mathsf{PGen}(1^\lambda)$$

We assume that $\mathsf{pp_{ch}}$ *is an implicit input to all other algorithms.*

KGen. *On input* $\mathsf{pp_{ch}}$*, it outputs the private and public key of the scheme:*

$$(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{KGen}(\mathsf{pp_{ch}})$$

Hash. *It gets as input a public key* $\mathsf{pk_{ch}}$ *and a message* m *to hash. It outputs a hash* h *and some randomness* r*:* $(h, r) \leftarrow \mathsf{Hash}(\mathsf{pk_{ch}}, m)$.[6]

Ver. *This deterministic algorithm gets as input the public key* $\mathsf{pk_{ch}}$*, a message* m*, randomness* r *and a hash* h*. It outputs a decision* $d \in \{0, 1\}$ *indicating whether* $(d = 1)$ *or not* $(d = 0)$ *the hash* h *is valid:*

$$d \leftarrow \mathsf{Ver}(\mathsf{pk_{ch}}, m, r, h)$$

Adap. *On input of secret key* $\mathsf{sk_{ch}}$*, the message* m*, the randomness* r*, hash* h *and a new message* m'*, it outputs new randomness* r'*:*

$$r' \leftarrow \mathsf{Adap}(\mathsf{sk_{ch}}, m, m', r, h)$$

Correctness. For a CH we require the correctness property to hold. In particular, we require that for all $\lambda \in \mathbb{N}$, for all $\mathsf{pp_{ch}} \leftarrow \mathsf{PGen}(1^\lambda)$, for all $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{KGen}(\mathsf{pp_{ch}})$, for all $m \in \mathcal{M}$, for all $(h, r) \leftarrow \mathsf{Hash}(\mathsf{pk_{ch}}, m)$, for all $m' \in \mathcal{M}$, we have for all for all $r' \leftarrow \mathsf{Adap}(\mathsf{sk_{ch}}, m, m', r, h)$, that $1 = \mathsf{Ver}(\mathsf{pk_{ch}}, m, r, h) = \mathsf{Ver}(\mathsf{pk_{ch}}, m', r', h)$. This definition captures perfect correctness.

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

Definition 2 (Indistinguishability). *A chameleon-hash* CH *is* indistinguishable*, if for any ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\left|\Pr[\mathsf{Ind}^{\mathsf{CH}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$. *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 is a stronger definition than key-exposure freeness [5, 27], as key-exposure freeness does *not* guarantee that for a given collision no additional ones can be found [26].

Definition 3 (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 randomnesses for the same message m^* such that the hashes are equal, for the same adversarially chosen pk^*.

Definition 4 (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 Figure 3.*

Definition 5 (Secure Chameleon-Hashes). *A chameleon-hash* CH *is secure, if it is correct, indistinguishable, and collision-resistant.*

It depends on the concrete use-case, if CH needs to be unique.

[6] The randomness r is also sometimes called "check value" [3].

```
Experiment $\mathsf{Ind}^{\mathsf{CH}}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp_{ch}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{KGen}(\mathsf{pp_{ch}})$
  $b \leftarrow \{0,1\}$
  $a \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt}(\mathsf{sk_{ch}},\cdot,\cdot,\cdot,b),\mathsf{Adap}(\mathsf{sk_{ch}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk_{ch}})$
    where $\mathsf{HashOrAdapt}$ on input $\mathsf{sk_{ch}}, m, m', b$:
      $(h, r) \leftarrow \mathsf{Hash}(\mathsf{pk_{ch}}, m')$
      $(h', r') \leftarrow \mathsf{Hash}(\mathsf{pk_{ch}}, m)$
      $r'' \leftarrow \mathsf{Adap}(\mathsf{sk_{ch}}, m, m', r', h')$
      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. Indistinguishability

```
Experiment $\mathsf{CollRes}^{\mathsf{CH}}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp_{ch}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{KGen}(\mathsf{pp_{ch}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{Adap}'(\mathsf{sk_{ch}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk_{ch}})$
    where $\mathsf{Adap}'$ on input $\mathsf{sk_{ch}}, m, m', r, h$:
      return $\bot$, if $\mathsf{Ver}(\mathsf{pk_{ch}}, m, r, h) \neq 1$
      $r' \leftarrow \mathsf{Adap}(\mathsf{sk_{ch}}, m, m', r, h)$
      return $\bot$, if $r' = \bot$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m, m'\}$
      return $r'$
  return 1, if $\mathsf{Ver}(\mathsf{pk_{ch}}, m^*, r^*, h^*) = 1 \ \wedge$
    $\mathsf{Ver}(\mathsf{pk_{ch}}, m'^*, r'^*, h^*) = 1 \ \wedge$
    $m'^* \notin \mathcal{Q} \ \wedge \ m^* \neq m'^*$
  return 0
```

Fig. 2. Collision Resistance

```
Experiment $\mathsf{Uniqueness}^{\mathsf{CH}}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp_{ch}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*) \leftarrow \mathcal{A}(\mathsf{pp_{ch}})$
  return 1, if $\mathsf{Ver}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{Ver}(\mathsf{pk}^*, m^*, r'^*, h^*) = 1 \ \wedge \ r^* \neq r'^*$
  return 0
```

Fig. 3. Uniqueness

3 CHs with Dual Long-Term Trapdoors

As already mentioned, a chameleon-hash with dual long-term trapdoors (CHDLTT) allows to prevent the holder of some long-term trapdoor $\mathsf{sk_{chret}}$ from finding collisions, as long as no additional second trapdoor std is known. This additional trapdoor can can be re-used for multiple hash generation, but may also be chosen freshly. This, e.g., allows to generate trapdoors in advance and to make one of the trapdoors public beforehand. Clearly, providing or withholding the second trapdoor information corresponding to the public part ptd of the second trapdoor thus allows to decide if finding a collision is possible for the holder of the long-term trapdoor. Due to this new possibility, we need to introduce a new framework, given subsequently, which is also accompanied by suitable security definitions. This framework is inspired by, and compatible to, the one given by Camenisch et al. [22] and it can be seen as a generalization of their ideas.

Definition 6 (CHDLTT)**.** *A chameleon-hash with dual long-term trapdoors* CHDLLT *is a tuple* $(\mathsf{PGen}, \mathsf{KGen}, \mathsf{TDGen}, \mathsf{Hash}, \mathsf{Ver}, \mathsf{Adap})$*, such that:*

PGen**.** *It outputs the public parameters:*

$$\mathsf{pp_{chret}} \leftarrow \mathsf{PGen}(1^\lambda)$$

which are input to all other algorithsm, which we sometimes may not make explicit for notational convenience.
KGen**.** *On input* $\mathsf{pp_{chret}}$*, it outputs the long-term private and public key of the scheme:*

$$(\mathsf{sk_{chret}}, \mathsf{pk_{chret}}) \leftarrow \mathsf{KGen}(\mathsf{pp_{chret}})$$

TDGen. *It outputs a private and public trapdoor pair:*

$$(\mathsf{std}, \mathsf{ptd}) \leftarrow \mathsf{TDGen}(\mathsf{pp}_{\mathsf{chret}})$$

Hash. *It gets as input a public key* $\mathsf{pk}_{\mathsf{chret}}$*,* ptd *and a message* m *to hash. It outputs a hash* h *and randomness* r*:*

$$(h, r) \leftarrow \mathsf{Hash}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}, m)$$

Ver. *It gets as input the public key* $\mathsf{pk}_{\mathsf{chret}}$*,* ptd*, a message* m*, a hash* h *and randomness* r*. It outputs a decision bit* $d \in \{0,1\}$*, indicating whether the given hash is correct:*

$$d \leftarrow \mathsf{Ver}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}, m, r, h)$$

Adap. *It gets as input* $\mathsf{sk}_{\mathsf{chret}}$*, the old message* m*, the old randomness* r*, the new message* m'*, the hash* h *and the trapdoor information* std*. It outputs new randomness:*

$$r' \leftarrow \mathsf{Adap}(\mathsf{sk}_{\mathsf{chret}}, m, m', r, h, \mathsf{std})$$

Correctness. For each CHDLTT we require the correctness properties to hold, i.e., for all security parameters $\lambda \in \mathbb{N}$, for all $\mathsf{pp}_{\mathsf{chret}} \leftarrow \mathsf{PGen}(1^\lambda)$, for all $(\mathsf{sk}_{\mathsf{chret}}, \mathsf{pk}_{\mathsf{chret}}) \leftarrow \mathsf{KGen}(\mathsf{pp}_{\mathsf{chret}})$, for all $(\mathsf{std}, \mathsf{ptd}) \leftarrow \mathsf{TDGen}(\mathsf{pp}_{\mathsf{chret}})$, for all $m \in \mathcal{M}$, for all $(h, r) \leftarrow \mathsf{Hash}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}, m)$, for all $m' \in \mathcal{M}$, for all $r' \leftarrow \mathsf{Adap}(\mathsf{sk}_{\mathsf{chret}}, m, m', r, h, \mathsf{std})$, it holds that $\mathsf{Ver}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}, m, r, h) = \mathsf{Ver}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}, m', r', h) = 1$. This definition captures perfect correctness. We also require some security guarantees, which we introduce next.

Indistinguishability. Indistinguishability requires that the randomnesses r does not reveal if it was obtained through Hash or Adap. In other words, an outsider cannot decide whether a message is the original one or not. Note, however, that both secrets are generated honestly, i.e., for adversarially generated keys no security guarantees are given. Moreover, only one trapdoor pair is generated; security for multiple trapdoor information pairs can be shown for any indistinguishable CHDLTT using a simple hybrid argument.

```
Experiment $\mathsf{Ind}_{\mathcal{A}}^{\mathsf{CHDLTT}}(\lambda)$
  $\mathsf{pp}_{\mathsf{chret}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{sk}_{\mathsf{chret}}, \mathsf{pk}_{\mathsf{chret}}) \leftarrow \mathsf{KGen}(\mathsf{pp}_{\mathsf{chret}})$
  $(\mathsf{std}, \mathsf{ptd}) \leftarrow \mathsf{TDGen}(\mathsf{pp}_{\mathsf{chret}})$
  $b \leftarrow \{0,1\}$
  $a \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt}(\mathsf{sk}_{\mathsf{chret}}, \mathsf{std}, \cdot, \cdot, b)}_{\mathsf{Adap}(\mathsf{sk}_{\mathsf{chret}}, \cdot, \cdot, \cdot, \cdot, \cdot)}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd})$
    where $\mathsf{HashOrAdapt}$ on input $\mathsf{sk}_{\mathsf{chret}}, \mathsf{std}, m, m', b$:
      let $(h, r) \leftarrow \mathsf{Hash}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}, m')$
      let $(h', r') \leftarrow \mathsf{Hash}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}, m)$
      let $r'' \leftarrow \mathsf{Adap}(\mathsf{sk}_{\mathsf{chret}}, m, m', r', h', \mathsf{std})$
      if $r'' = \bot \lor r' = \bot$, return $\bot$
      if $b = 0$, return $(h, r)$
      if $b = 1$, return $(h', r'')$
  return 1, if $a = b$
  return 0
```

Fig. 4. Indistinguishability

```
Experiment $\mathsf{Uniqueness}_{\mathcal{A}}^{\mathsf{CHDLTT}}(\lambda)$
  $\mathsf{pp}_{\mathsf{chret}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{pk}^*, m^*, r^*, r'^*, \mathsf{ptd}^*, h^*) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{ch}})$
  return 1, if $\mathsf{Ver}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}^*, m^*, r^*, h^*) =$
    $\mathsf{Ver}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}^*, m^*, r'^*, h^*) = 1 \land r^* \neq r'^*$
  return 0
```

Fig. 5. Uniquessness

Definition 7 (Indistinguishability). *A* CHDLTT *is* indistinguishable*, if for every ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\left|\Pr[\mathsf{Ind}_{\mathcal{A}}^{\mathsf{CHDLTT}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$ *. The corresponding experiment is depicted in Fig. 4.*

Public Collision Resistance. Public collision resistance requires that, even if an adversary has access to an Adap oracle, it cannot find any collisions by itself, even if it can chose ptd. Clearly, the collision must be fresh, i.e., must not be produced using the Adap oracle.

Definition 8 (Public Collision-Resistance). *A* CHDLTT *is publicly collision-resistant, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{PublicCollRes}_{\mathcal{A}}^{\mathsf{CHDLTT}}(1^\lambda) = 1] \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 6.*

Private Collision-Resistance. Private collision-resistance requires that even the holder of the secret key $\mathsf{sk}_{\mathsf{chret}}$ cannot find collisions as long as std is unknown, even if it can request collisions for different pks. This catches the idea that the hashes for a given ptd may be equivocated for different $\mathsf{pk}_{\mathsf{chret}}$s. This is formalized by a honest adaption oracle which returns collisions for other key pairs. Hence, $\mathcal{A}$'s goal is to return an actual collision for a public key pk^*, for an honestly generated ptd, for which it did not see a collision generated by the oracle, but for an arbitrary $\mathsf{sk}_{\mathsf{chret}}$.

```
Experiment $\mathsf{PublicCollRes}_{\mathcal{A}}^{\mathsf{CHDLTT}}(\lambda)$
  $\mathsf{pp}_{\mathsf{chret}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{sk}_{\mathsf{chret}}, \mathsf{pk}_{\mathsf{chret}}) \leftarrow \mathsf{KGen}(\mathsf{pp}_{\mathsf{chret}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, r^*, m'^*, r'^*, \mathsf{ptd}^*, h^*) \leftarrow \mathcal{A}^{\mathsf{Adap}'(\mathsf{sk}_{\mathsf{chret}}, \cdots)}(\mathsf{pk}_{\mathsf{chret}})$
    where $\mathsf{Adap}'$ on input $\mathsf{sk}_{\mathsf{chret}}, m, m', r, \mathsf{std}, \mathsf{ptd}, h$:
      return $\perp$, if $\mathsf{Ver}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}, m, r, h) = 0$
      $r' \leftarrow \mathsf{Adap}(\mathsf{sk}_{\mathsf{chret}}, m, m', r, h, \mathsf{std})$
      If $r' = \perp$, return $\perp$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{ptd}, m), (\mathsf{ptd}, m')\}$
      return $r'$
  return 1, if $\mathsf{Ver}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}^*, m^*, r^*, h^*) = 1 \wedge$
    $\mathsf{Ver}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}^*, m'^*, r'^*, h^*) = 1 \wedge$
    $(\mathsf{ptd}^*, m'^*) \notin \mathcal{Q} \ \wedge \ m^* \neq m'^*$
  return 0
```

Fig. 6. Public Collision-Resistance

```
Experiment $\mathsf{PrivateCollRes}_{\mathcal{A}}^{\mathsf{CHDLTT}}(\lambda)$
  $\mathsf{pp}_{\mathsf{chret}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{std}, \mathsf{ptd}) \leftarrow \mathsf{TDGen}(\mathsf{pp}_{\mathsf{chret}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(\mathsf{pk}^*, m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{Adap}'(\mathsf{std}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot)}(\mathsf{ptd})$
    where $\mathsf{Adap}'$ on input $\mathsf{std}$, $\mathsf{sk}_{\mathsf{chret}}$,
    $\mathsf{pk}_{\mathsf{chret}}, m, m', r, h$:
      return $\perp$, if $\mathsf{Ver}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}, m, r, h) = 0$
      $r' \leftarrow \mathsf{Adap}(\mathsf{sk}_{\mathsf{chret}}, m, m', r, h, \mathsf{std})$
      return $\perp$, if $r' = \perp$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}_{\mathsf{chret}}, m), (\mathsf{pk}_{\mathsf{chret}}, m')\}$
      return $r'$
  return 1, if $\mathsf{Ver}(\mathsf{pk}^*, \mathsf{ptd}, m^*, r^*, h^*) = 1 \wedge$
    $\mathsf{Ver}(\mathsf{pk}^*, \mathsf{ptd}, m'^*, r'^*, h^*) = 1 \wedge$
    $(\mathsf{pk}^*, m'^*) \notin \mathcal{Q} \ \wedge \ m^* \neq m'^*$
  return 0
```

Fig. 7. Private Collision-Resistance

Definition 9 (Private Collision-Resistance). *A* CHDLTT *is privately collision-resistant, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{PrivateCollRes}_{\mathcal{A}}^{\mathsf{CHDLTT}}(1^\lambda) = 1] \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 7.*

Uniqueness. Uniquess requires that even if the adversary can generate the public key and the corresponding public trapdoor, it cannot find two different randomnesses for the same hash and message such that both are valid.

Definition 10 (Uniqueness). *A* CHDLTT *is unique, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Uniqueness}_{\mathcal{A}}^{\mathsf{CHDLTT}}(1^\lambda) = 1] \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 5.*

Definition 11 (Secure CHDLTT). *We call a* CHDLTT *secure, if it is correct, indistinguishable, publicly collision-resistant and privately collision-resistant.*

As for CHs, it depends on the concrete use-case, if uniqueness is required.

$\mathsf{PGen}(1^\lambda)$**.** Return $\mathsf{pp_{chret}} \leftarrow \mathsf{CH.PGen}(1^\lambda)$.
$\mathsf{KGen}(\mathsf{pp_{chret}})$**.** Return $(\mathsf{sk_{chret}}, \mathsf{pk_{chret}}) \leftarrow \mathsf{CH.KGen}(\mathsf{pp_{chret}})$.
$\mathsf{TDGen}(\mathsf{pp_{chret}})$**.** Return $(\mathsf{std}, \mathsf{ptd}) \leftarrow \mathsf{CH.KGen}(\mathsf{pp_{chret}})$.

$\mathsf{Hash}(\mathsf{pk_{chret}}, \mathsf{ptd}, m)$**.** Let $(h_1, r_1) \leftarrow \mathsf{CH.Hash}(\mathsf{pk_{chret}}, (\mathsf{pk_{chret}}, \mathsf{ptd}, m))$ and $(h_2, r_2) \leftarrow \mathsf{CH.Hash}(\mathsf{ptd}, (\mathsf{pk_{chret}}, \mathsf{ptd}, m))$. Return $((h_1, h_2), (r_1, r_2))$.

$\mathsf{Ver}(\mathsf{pk_{chret}}, \mathsf{ptd}, m, r, h)$**.** Let $b_1 \leftarrow \mathsf{CH.Ver}(\mathsf{pk_{chret}}, (\mathsf{pk_{chret}}, \mathsf{ptd}, m), r_1, h_1)$ and $b_2 \leftarrow \mathsf{CH.Ver}(\mathsf{ptd}, (\mathsf{pk_{chret}}, \mathsf{ptd}, m), r_2, h_2)$. If $b_1 = 0 \lor b_2 = 0$, return 0. Return 1.

$\mathsf{Adap}(\mathsf{sk_{chret}}, m, m', r, h, \mathsf{std})$**.** If $0 = \mathsf{Ver}(\mathsf{pk_{ch}}, m, r, h)$, return $\bot$. Compute $r'_1 \leftarrow \mathsf{CH.Adap}(\mathsf{sk_{chret}}, (\mathsf{pk_{chret}}, \mathsf{ptd}, m), (\mathsf{pk_{chret}}, \mathsf{ptd}, m'), r_1, h_1)$, and $r'_2 \leftarrow \mathsf{CH.Adap}(\mathsf{std}, (\mathsf{pk_{chret}}, \mathsf{ptd}, m), (\mathsf{pk_{chret}}, \mathsf{ptd}, m'), r_2, h_2)$. Return (r'_1, r'_2).

Construction 1: Black-box construction of CHDLTT

4 Constructions

We first show how to bootstrap a CHDLTT scheme in a black-box fashion from any given existing secure chameleon-hash CH, inspired by the ideas given by Camenisch et al. [22] who also proposed a suitable chameleon-hash CH based on the ideas by Brzuska et al. [17]. We then present a direct construction in the hidden order group setting based on the one-more RSA-Assumption.

Black-Box Construction. We now present a black-box construction from any existing chameleon-hash CH. Namely, we show how one can achieve our goals by combining two instances of a secure chameleon-hash CH. However, instead of hashing the messages alone, one also requires to hash the public keys $\mathsf{pk_{chret}}$ and ptd in question to achieve our strengthened definitions. This is described in detail in Construction 1.

Theorem 1. *If* CH *is secure (and unique), then Construction 1 is a secure (and unique)* CHDLTT*.*

We sketch the proof below. The detailed proof of Theorem 1 is given in App. B.

Proof (Sketch). Indistinguishability follows from the indistinguishability of the underlying chameleon-hashes. If an adversary can find a collision (either for public and private collision-resistance), than either h_1 or h_2 must be a fresh collision, which can easily be extracted. The need to also hash the public keys comes from the fact that the adversary is, in our model, allowed to choose the trapdoor. For uniqueness, the adversary must find two randomness values for the same hash and message. Thus, the randomness for either h_1 or h_2 must be non-unique.

It remains to show if we can also directly construct a CHDLTT, which we answer to the affirmative subsequently.

A Direct Construction. We now present a direct construction, based on the one-more RSA-Assumption. It is inspired by ideas due to Brzuska et al. [17], Pöhls et al. [53] and Camenisch et al. [22], enriched with the trick to also hash the public keys, as introduced in our first construction.

In our construction (illustrated in Construction 2), the public trapdoor ptd is an additional RSA-modulus n'. Thus, only if the factorization of $n' = p'q'$, contained in std, and $n = pq$, which is the secret key $\mathsf{sk_{ch}}$, is known, a collision can be produced. We assume that the bit-length of n and n' is the same, which is implicitly given by the security parameter λ. Furthermore, we assume that an algorithm aborts if a modulus is too large for the given security parameter. We note that the condition on the size of e also implies $\gcd(\varphi(nn'), e) = 1$, which makes the construction a bit more efficient than the solution by Camenisch et al. [22] which require $e > n^3$. Let $\mathcal{H}_n : \{0,1\}^* \to \mathbb{Z}_n^*$, where $n \in \mathbb{N}$, denote a random oracle.

Theorem 2. *If the one-more RSA-Assumption holds, then Construction 2 is a secure* CHDLTT *in the random-oracle model.*

$\mathsf{PGen}(1^\lambda)$**.** Call RSA with the restriction that e is larger than any possible n w.r.t λ and e prime. Return e.
$\mathsf{KGen}(\mathsf{pp}_{\mathsf{chret}})$**.** Generate two primes p and q using $\mathsf{RSA}(1^\lambda)$. Set $\mathsf{sk}_{\mathsf{ch}} \leftarrow (p,q)$. Let $n \leftarrow pq$. Set $\mathsf{pk}_{\mathsf{ch}} \leftarrow n$. Return $(\mathsf{sk}_{\mathsf{chret}}, \mathsf{pk}_{\mathsf{chret}})$.
$\mathsf{TDGen}(\mathsf{pp}_{\mathsf{chret}})$**.** Generate two primes p' and q' using $\mathsf{RSA}(1^\lambda)$. Set $\mathsf{std} \leftarrow (p',q')$, $n' \leftarrow p'q'$, and $\mathsf{ptd} \leftarrow n'$. If $\gcd(n,n') \neq 1$, start over. Return $(\mathsf{std}, \mathsf{ptd})$.

$\mathsf{Hash}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}, m)$**.** Draw $r \leftarrow \mathbb{Z}^*_{nn'}$. Let $g \leftarrow \mathcal{H}_{nn'}(m,n,n')$ and $h \leftarrow gr^e \bmod nn'$. Return (h,r).
$\mathsf{Ver}(\mathsf{pk}_{\mathsf{chret}}, \mathsf{ptd}, m, r, h)$**.** Return $\perp$, if $r \notin \mathbb{Z}^*_{nn'}$. Let $g \leftarrow \mathcal{H}_{nn'}(m,n,n')$ and $h' \leftarrow gr^e \bmod nn'$. Return 1, if $h = h'$. Return 0.
$\mathsf{Adap}(\mathsf{sk}_{\mathsf{chret}}, m, m', r, h, \mathsf{std})$**.** Check that $n' = p'q'$, where p' and q' are taken from std. If this is not the case, return $\perp$. If $\mathsf{Ver}(\mathsf{pk}_{\mathsf{ch}}, m, r, h) = 0$, return $\perp$. Let d s.t. $de \equiv 1 \bmod \varphi(nn')$, $g \leftarrow \mathcal{H}_{nn'}(m,n,n')$, $h \leftarrow gr^e \bmod nn'$, $g' \leftarrow \mathcal{H}_{nn'}(m',n,n')$ and $r' \leftarrow (h(g'^{-1}))^d \bmod nn'$. Return r'.

Construction 2: CHDLTT from the one-more RSA-Assumption

Table 1. Percentiles for key generation in ms

	KGen	KGen_2	TDGen	TDGen_2
Min.	45	43	45	43
25%	107	112	111	109
Med.	164	168	164	163
75%	239	240	238	243
90%	331	319	320	329
95%	383	374	368	390
Max.	525	505	533	553
Avg.	186	186	184	187
SD	99.34	96.00	95.50	101.30

Table 2. Percentiles for the other algorithms in ms

	Hash	Hash_2	Ver	Ver_2	Adap	Adap_2
Min.	20	37	20	37	53	121
25%	22	39	22	38	70	125
Med.	23	40	23	39	75	130
75%	28	43	27	43	85	139
90%	35	49	33	49	106	154
95%	45	54	43	54	138	164
Max.	137	107	153	91	399	258
Avg.	27	42	27	42	85	135
SD	12.75	6.73	12.36	6.21	35.32	14.93

We sketch the proof below. The detailed proof of Theorem 1 is given in App. C.

Proof (Sketch). Indistinguishability follows from the fact that a random oracle behaves as a function, while RSA defines a permutation with the restrictions given and the values are distributed uniformly in $\mathbb{Z}^*_{nn'}$. If an adversary can find a collision (either for public or private collision-resistance), than a reduction can extract an e^{th} root of one of the moduli, while uniqueness follows from the same fact used for indistinguishability.

5 Evaluation

To demonstrate the practicality of our schemes, we have implemented them in Java. We implemented both CHDLTT construction, whereas for the first we use the standard RSA-based chameleon-hash as the underlying primitive. All RSA-moduli have a fixed bit-length of 2,048 Bit (with balanced primes). Likewise, e has 2,050 Bit. The random oracles were implemented using SHA-512 in standard counter-mode [11, 23, 37].

The measurements were performed on a Lenovo W530 with an Intel i7-3470QM@2.70 Ghz, and 16 GiB of RAM. No performance optimization such as CRT were implemented and the computation is performed by a single thread. The reason for this choice is to obtain a lower bound of the runtime and any additional optimization will only speed up the calculation. An overview is given in Fig. 8, containing the averages of each algorithm. More detailed results are depicted in Fig. 9, Fig. 10, Tab. 1 and Tab. 2. In the figures, and tables, the algorithms of Construction 1 have no subscript, while the algorithms of Construction 2 are sub-scripted with “2”. 1,000 runs were taken. Parameter generation is omitted, as this is a one-time setup, i.e., drawing a random prime.
As demonstrated by the runtime measurements, both primitives can be considered practical. The lion’s share is finding suitable primes during key generation. All exponentiations within the algorithms only have a negligible

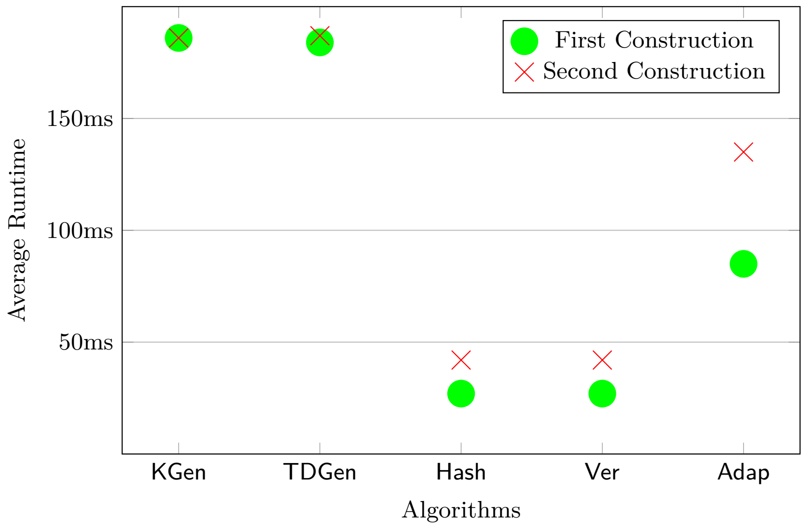

Fig. 8. Overview of our measurements in ms

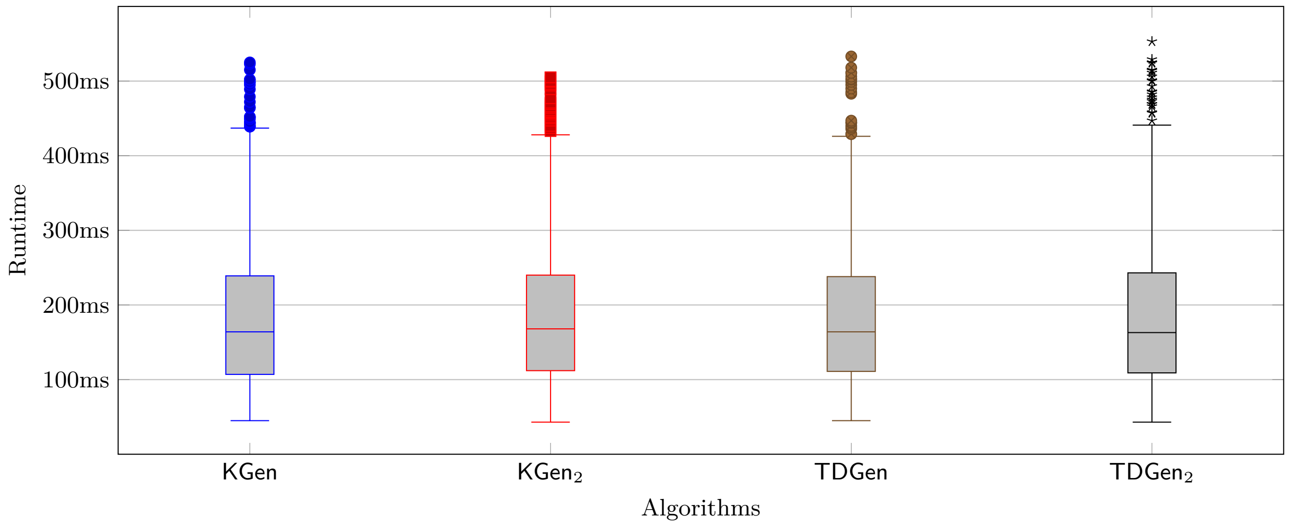

Fig. 9. Key generation algorithm runtime in ms

overhead, as seen by the runtime. However, as key generation only needs to be done once, this seems to be acceptable, even with realistic security parameters and rather expensive RSA-based primitives.

6 Application: Three-Party Sanitizable Signatures 3SSS

We now present applications of our CHDLTTs. The first are *three-party* sanitizable signatures. Sanitizable signatures (SSS) allow a signer to determine which blocks $m[i]$ of a message $m = (m[1], m[2], \ldots, m[i], \ldots, m[\ell])$

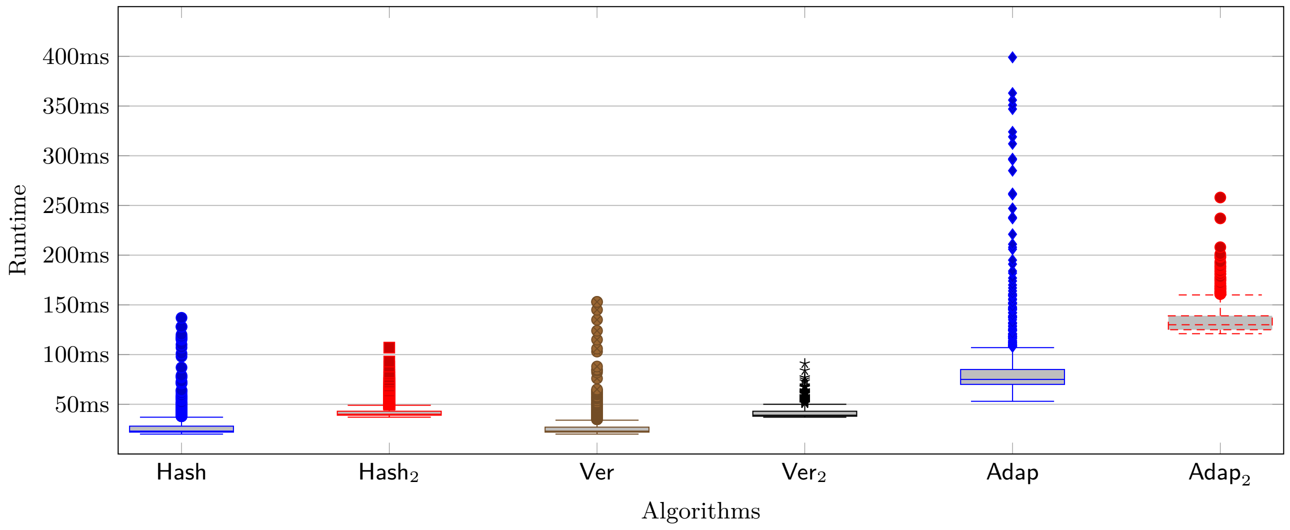

Fig. 10. Other algorithms' runtime in ms

are admissible. Any such admissible block can be changed to an arbitrary bitstring $m[i]' \in \{0,1\}^*$, where $i \in \{1, 2, \ldots, \ell\}$, by a semi-trusted party named the sanitizer, if it had received an additional secret. In a nutshell, 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 for m' under the original public key. Sanitizable signatures enforce that either a block cannot be altered at all, or only by a semi-trusted third party. In a 3SSS, the signer additionally appoints a third party which needs to provide its secret key before the sanitizer can alter certain blocks.

Related Work and State-of-the-Art for SSS. Sanitizable signatures have been introduced by Ateniese et al. [2]. Most of the current security properties were formalized by Brzuska et al. [17]. Later on, extensions such as (strong) unlinkability [19, 21, 36] and non-interactive public accountability [20, 21] were introduced. Additional extensions such as limiting the sanitizer to certain values [24, 32, 46, 55], multi-sanitizer and multi-signer environments [18, 21, 25], as well as sanitization of signed encrypted data [30, 34] have been considered. They have also been used as a tool to make other primitives accountable [54] and to construct other primitives, such as redactable signatures [12, 50]. We stress that there already exists work in the multiple signer/sanitizer setting [18, 21, 25], but in their case each sanitizer holds its own key and thus each sanitizer must be involved for each sanitization. In contrast, in our case, one key is given to the sanitizer once sanitization should be made possible. Thus, exactly one party can then sanitize as often as it wants.[7] More detailed overviews of this field are already published [13, 31].

The Framework for Three-Party Sanitizable Signature Schemes. Subsequently, we introduce the used framework for 3SSSs. The definitions are inspired by the ones given by Camenisch et al. [22], which are itself based on existing work [2, 17, 20, 21, 41, 48]. However, due to our goals, we need to re-define the framework. Like Camenisch et al., we do not consider "non-interactive public accountability" [20, 21, 45], which allows some

[7] In the previous version of this paper this claim was formulated misleading. We hope that this formulation is clearer to the reader.

external party to decide which party is accountable, as transparency is mutually exclusive to this property. But if needed this is very easy to achieve, e.g., by signing the signature again [20].

For brevity, we now set some additional notation. This notation is similar to existing definitions, to make reading more comfortable [17, 22]. 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. The variable ADM is a tuple containing the set of indices of the admissible blocks and the number ℓ of blocks in a message m. We write $\mathrm{ADM}(m) = 1$, if ADM is valid w.r.t. m, i.e., ADM contains the correct ℓ and all indices are in m. For example, let $\mathrm{ADM} = (\{1, 2, 4\}, 4)$. Then, m must contain four blocks, while all but the third will be admissible. If we write $m_i \in \mathrm{ADM}$, we mean that m_i is admissible. MOD is a set containing pairs $(i, m[i]')$ for those blocks that shall be modified, meaning that $m[i]$ is replaced with $m[i]'$. We write $\mathrm{MOD}(\mathrm{ADM}) = 1$, if MOD is valid w.r.t. ADM, meaning that the indices to be modified are contained in ADM. To allow 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 (Three-Party Sanitizable Signature). *A three-party sanitizable signature scheme* 3SSS *consists of nine ppt algorithms* (PGen, $\mathsf{KGen_{sig}}$, $\mathsf{KGen_{san}}$, TGen, Sign, Sanit, Verify, Proof, Judge) *such that:*

PGen. *On input security parameter λ, it generates the public parameters:*

$$\mathsf{pp_{3SSS}} \leftarrow \mathsf{PGen}(1^\lambda)$$

We assume that $\mathsf{pp_{3SSS}}$ *is implicitly input to all other algorithms.*

$\mathsf{KGen_{sig}}$. *It takes the public parameters* $\mathsf{pp_{3SSS}}$ *and returns the signer's private key and the corresponding public key:*

$$(\mathsf{sk_{sig}}, \mathsf{pk_{sig}}) \leftarrow \mathsf{KGen_{sig}}(\mathsf{pp_{3SSS}})$$

$\mathsf{KGen_{san}}$. *It takes the public parameters* $\mathsf{pp_{3SSS}}$ *and returns the sanitizer's private key and the corresponding public key:*

$$(\mathsf{sk_{san}}, \mathsf{pk_{san}}) \leftarrow \mathsf{KGen_{san}}(\mathsf{pp_{3SSS}})$$

TGen. *It takes the public parameters* $\mathsf{pp_{3SSS}}$*, and returns the private and public trapdoor pair:*

$$(\mathsf{tdpriv}, \mathsf{tdpub}) \leftarrow \mathsf{TGen}(\mathsf{pp_{3SSS}})$$

Sign. *It takes as input a message m,* $\mathsf{sk_{sig}}$*,* $\mathsf{pk_{san}}$*,* tdpub*, as well as a description* ADM *of the admissible blocks. If* $\mathrm{ADM}(m) = 0$*, it returns* $\perp$*. It outputs a signature:*

$$\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk_{sig}}, \mathsf{pk_{san}}, \mathsf{tdpub}, \mathrm{ADM})$$

Sanit. *It takes a message m, modification instruction* MOD*, a signature σ,* $\mathsf{pk_{sig}}$*,* $\mathsf{sk_{san}}$ *and* tdpriv*. It outputs m' together with σ':*

$$(m', \sigma') \leftarrow \mathsf{Sanit}(m, \mathrm{MOD}, \sigma, \mathsf{pk_{sig}}, \mathsf{sk_{san}}, \mathsf{tdpriv})$$

where $m' \leftarrow \mathrm{MOD}(m)$ *is message m modified according to the modification instruction* MOD*. If* $\mathrm{ADM}(m') = 0$*, this algorithm returns* $\perp$*.*

Verify. *It takes as input the signature σ for a message m w.r.t. the public keys* $\mathsf{pk_{sig}}$*,* $\mathsf{pk_{san}}$*, and* tdpub*. It outputs a decision* $d \in \{1, 0\}$*:*

$$d \leftarrow \mathsf{Verify}(m, \sigma, \mathsf{pk_{sig}}, \mathsf{pk_{san}}, \mathsf{tdpub})$$

This algorithm is deterministic.

Proof. *It takes as input* $\mathsf{sk_{sig}}$*, a message m, a signature σ, a set of polynomially many additional message/signature pairs* $\{(m_i, \sigma_i)\}$*,* $\mathsf{pk_{san}}$ *and* tdpub*. 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)*:*

$$\pi \leftarrow \mathsf{Proof}(\mathsf{sk_{sig}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \mathsf{pk_{san}}, \mathsf{tdpub})$$

```
**Experiment** $\mathsf{Unforgeability}_{\mathcal{A}}^{\mathsf{3SSS}}(\lambda)$
  $\mathsf{pp}_{\mathsf{3SSS}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{3SSS}})$
  $(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) \leftarrow \mathsf{KGen}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{3SSS}})$
  $(m^*, \sigma^*, \mathsf{tdpub}^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, \mathsf{sk}_{\mathsf{sig}}, \cdot, \cdot, \cdot), \mathsf{Sanit}(\cdot, \cdot, \cdot, \cdot, \mathsf{sk}_{\mathsf{san}}, \cdot)}_{\mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}}, \cdot, \cdot, \cdot, \cdot, \cdot)}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$
    for $i = 1, \ldots, q$ let $(m_i, \mathsf{pk}_{\mathsf{san},i}, \mathsf{tdpub}_i, \mathrm{ADM}_i)$, and $\sigma_i$,
      index the queries/answers to/from $\mathsf{Sign}$
    for $j = 1, \ldots, q'$ let $(m_j, \sigma_j, \mathsf{pk}_{\mathsf{sig},j}, \mathrm{MOD}_j, \mathsf{tdpriv}_i)$,
      and $(m'_j, \sigma'_j)$, index the queries/answers to/from $\mathsf{Sanit}$
  return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \mathsf{tdpub}^*) = 1 \wedge$
    $\forall i \in \{1, \ldots, q\} : (\mathsf{pk}_{\mathsf{san}}, m^*, \mathsf{tdpub}^*) \neq (\mathsf{pk}_{\mathsf{san},i}, m_i, \mathsf{tdpub}_i) \wedge$
    $\forall j \in \{1, \ldots, q'\} : (\mathsf{pk}_{\mathsf{sig}}, m^*, \mathsf{tdpub}^*) \neq (\mathsf{pk}_{\mathsf{sig},j}, m'_j, \mathsf{tdpub}'_j)$
  return 0
```

Fig. 11. Unforgeability

Judge. *It takes as input a message m, a signature σ, $\mathsf{pk}_{\mathsf{sig}}$, $\mathsf{pk}_{\mathsf{san}}$, tdpub, 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 $d \in \{\mathsf{Sig}, \mathsf{San}, \perp\}$, indicating whether the message/signature pair has been created by the signer, or the sanitizer:*

$$d \leftarrow \mathsf{Judge}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \mathsf{tdpub}, \pi)$$

Defining correctness of 3SSS is straightforward and therefore omitted.

Security of 3SSSs. Next, we introduce the security model. It is based on the work done for sanitizable signatures by Brzuska et al. [17]. However, due to our goals, we need to modify it significantly. In the security framework (for the unforgeability definitions), the public trapdoor tdpub is always generated by the adversary, as the holder of tdpriv should never be able to generate forgeries. This simplifies our framework significantly, as an adversary generating tdpub has more power than an adversary only receiving an honestly generated tdpub.

Unforgeability. This definition requires that an adversary $\mathcal{A}$ not having any secret keys is not able to produce a valid signature σ^* on a *new* message m^*. As $\mathcal{A}$ has full oracle access *new* means that $\mathcal{A}$ has not seen the message genuinely being signed before.

Definition 13 (Unforgeability). *An 3SSS is unforgeable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Unforgeability}_{\mathcal{A}}^{\mathsf{3SSS}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 11.*

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 3SSS is immutable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Immutability}_{\mathcal{A}}^{\mathsf{3SSS}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 12.*

```
$\textbf{Experiment}\ \mathsf{Immutability}^{\mathsf{3SSS}}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp_{3SSS}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{sk_{sig}}, \mathsf{pk_{sig}}) \leftarrow \mathsf{KGen_{sig}}(\mathsf{pp_{3SSS}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{tdpub}^*) \leftarrow \mathcal{A}^{\mathsf{Sign}'(\cdot, \mathsf{sk_{sig}}, \cdot, \cdot, \cdot), \mathsf{Proof}(\mathsf{sk_{sig}}, \cdot, \cdot, \cdot, \cdot, \cdot)}(\mathsf{pk_{sig}})$
    where $\mathsf{Sign}'$ on input $\mathsf{pk_{san}}$, $\mathsf{sk_{sig}}$, $m$, ADM, $\mathsf{tdpub}$:
      for all $m'_i \in \{\mathrm{MOD}(m_i) \mid \mathrm{MOD}$ with $\mathrm{MOD}(\mathrm{ADM}_i) = 1\})$
        let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk_{san}}, \mathsf{tdpub}, m'_i)\}$
      return $\mathsf{Sign}(m, \mathsf{sk_{sig}}, \mathsf{pk_{san}}, \mathsf{tdpub}, \mathrm{ADM})$
  return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk_{sig}}, \mathsf{pk}^*, \mathsf{tdpub}^*) = 1\ \wedge$
    $(\mathsf{pk}^*, \mathsf{tdpub}^*, m^*) \notin \mathcal{Q}$
  return 0
```

Fig. 12. Immutability

```
$\textbf{Experiment}\ \mathsf{Immutability2}^{\mathsf{3SSS}}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp_{3SSS}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{tdpriv}, \mathsf{tdpub}) \leftarrow \mathsf{TDGen}(\mathsf{pp_{chret}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, \sigma^*, \mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}, \pi^*) \leftarrow \mathcal{A}^{\mathsf{Sanit}'(\cdot, \cdot, \cdot, \cdot, \cdot, \mathsf{tdpriv})}(\mathsf{tdpub})$
    where $\mathsf{Sanit}'$ on input $m$, MOD, $\sigma$, $\mathsf{pk_{sig}}$, $\mathsf{sk_{san}}$, $\mathsf{tdpriv}$:
      let $(m', \sigma') \leftarrow \mathsf{Sanit}(m, \mathrm{MOD}, \sigma, \mathsf{pk_{sig}}, \mathsf{sk_{san}}, \mathsf{tdpriv})$
      return $\perp$, if $\mathsf{Verify}(m', \sigma', \mathsf{pk_{sig}}, \mathsf{pk_{san}}, \mathsf{tdpub}) = 0$
      let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk_{sig}}, \mathsf{pk_{san}}, m), (\mathsf{pk_{sig}}, \mathsf{pk_{san}}, m')\}$
      return $\mathsf{Sign}(m, \mathsf{sk_{sig}}, \mathsf{pk_{san}}, \mathsf{tdpub}, \mathrm{ADM})$
  return 1, if $\mathsf{Judge}(m^*, \sigma^*, \mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}, \mathsf{tdpub}, \pi^*) = \mathsf{San}\ \wedge$
    $(\mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}, m^*) \notin \mathcal{Q}$
  return 0
```

Fig. 13. Immutability2

Immutability2. Even if the signer, and the sanitizer, work together, they must be not able to generate a valid sanitization, if std is not known. However, as the signer can clearly sign whatever it wants (including any ADM), we define this immutability property such that the adversary cannot generate a proof π^* that it sanitized the message, as it should not be able to do so. The adversary also receives full adaptive oracle access, also for other pks, modeling the case where for different pks sanitizations where generated.

Definition 15 (Immutability2). *An* 3SSS *is immutable2, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Immutability2}^{\mathsf{3SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 13.*

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

```
**Experiment** $\mathsf{Privacy}_{\mathcal{A}}^{\mathsf{3SSS}}(\lambda)$
  $\mathsf{pp}_{\mathsf{3SSS}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{3SSS}})$
  $(\mathsf{sk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{san}}) \leftarrow \mathsf{KGen}_{\mathsf{san}}(\mathsf{pp}_{\mathsf{3SSS}})$
  $(\mathsf{tdpriv}, \mathsf{tdpub}) \leftarrow \mathsf{TGen}(\mathsf{pp}_{\mathsf{3SSS}})$
  $b \leftarrow \{0,1\}$
  $a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{san}},\mathsf{tdpriv}),}_{\mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{LoRSanit}(\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{sk}_{\mathsf{sig}},\mathsf{sk}_{\mathsf{san}},\mathsf{tdpriv},b)}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \mathsf{tdpub})$
    where $\mathsf{LoRSanit}$ on input of
      $m_0$, $m_1$, $\mathrm{MOD}_0$, $\mathrm{MOD}_1$, ADM, $\mathsf{sk}_{\mathsf{sig}}$, $\mathsf{sk}_{\mathsf{san}}$, $\mathsf{tdpriv}$, $b$
      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 $\sigma \leftarrow \mathsf{Sign}(m_b, \mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}}, \mathsf{tdpub}, \mathrm{ADM})$
      return $(m', \sigma') \leftarrow \mathsf{Sanit}(m_b, \mathrm{MOD}_b, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{san}}, \mathsf{tdpriv})$
  return 1, if $a = b$
  return 0
```

Fig. 14. Privacy

has to decide which of the input messages was used to generate the sanitized one. The adversary receives full adaptive oracle access.

Definition 16 (Privacy). *An* 3SSS *is called private, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\left|\Pr[\mathsf{Privacy}_{\mathcal{A}}^{\mathsf{3SSS}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 14.*

Transparency. Transparency guarantees that the accountable party of a message m remains anonymous. This is important if discrimination may follow from learning that sanitizations have taken place [2, 17]. In a nutshell, the adversary to transparency has to decide whether it sees a freshly computed signature, or a sanitized one. The adversary has full (but proof-restricted) oracle access.

Definition 17 ((Proof-Restricted) Transparency). *An* 3SSS *is proof-restricted transparent, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\left|\Pr[\mathsf{Transparency}_{\mathcal{A}}^{\mathsf{3SSS}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$, *where the corresponding experiment is defined in Fig. 15.*

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

Signer-Accountability. For signer-accountability, a signer must not be able to blame a sanitizer if the sanitizer is actually not responsible for a given message. 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 m^* output by $\mathcal{A}$. Here, the adversary gains access to all oracles related to sanitizing.

Definition 18 (Signer-Accountability). *An* 3SSS *is signer-accountable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Sig\text{-}Accountability}_{\mathcal{A}}^{\mathsf{3SSS}}(\lambda) = 1] \leq \nu(\lambda)$, *where the experiment is defined in Fig. 16.*

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

```
**Experiment** $\mathsf{Transparency}_{\mathcal{A}}^{\mathsf{3SSS}}(\lambda)$
  $\mathsf{pp_{3SSS}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{sk_{sig}}, \mathsf{pk_{sig}}) \leftarrow \mathsf{KGen_{sig}}(\mathsf{pp_{3SSS}})$
  $(\mathsf{sk_{san}}, \mathsf{pk_{san}}) \leftarrow \mathsf{KGen_{san}}(\mathsf{pp_{3SSS}})$
  $(\mathsf{tdpriv}, \mathsf{tdpub}) \leftarrow \mathsf{TGen}(\mathsf{pp_{3SSS}})$
  $b \leftarrow \{0,1\}$
  $\mathcal{Q} \leftarrow \emptyset$
  $a \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk_{sig}},\cdot,\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{sk_{san}}),\mathsf{Proof}'(\mathsf{sk_{sig}},\cdot,\cdot,\cdot,\cdot,\cdot)}_{\mathsf{Sanit/Sign}(\cdot,\cdot,\cdot,\mathsf{sk_{sig}},\mathsf{sk_{san}},\mathsf{tdpriv},b)}(\mathsf{pk_{sig}}, \mathsf{pk_{san}})$
    where $\mathsf{Proof}'$ on input of
      $\mathsf{sk_{sig}}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \mathsf{pk}'_{\mathsf{san}}, \mathsf{tdpub}, b$:
      return $\perp$, if $\mathsf{pk}'_{\mathsf{san}} = \mathsf{pk_{san}} \ \wedge\ (m \in \mathcal{Q} \ \vee\ \exists (m_j, \sigma_j) \in \{(m_i, \sigma_i) \mid i \in \mathbb{N}\} : m_j \in \mathcal{Q})$
      return $\mathsf{Proof}(\mathsf{sk_{sig}}, m, \sigma, \{(m_i, \sigma_i)\}, \mathsf{pk}'_{\mathsf{san}}, \mathsf{tdpub})$
    where $\mathsf{Sanit/Sign}$ on input of
      $m, \mathrm{MOD}, \mathrm{ADM}, \mathsf{sk_{sig}}, \mathsf{sk_{san}}, \mathsf{tdpriv}, b$:
      $\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk_{sig}}, \mathsf{pk_{san}}, \mathsf{tdpub}, \mathrm{ADM})$
      $(m', \sigma') \leftarrow \mathsf{Sanit}(m, \mathrm{MOD}, \sigma, \mathsf{pk_{sig}}, \mathsf{sk_{san}}, \mathsf{tdpriv})$
      if $b = 1$:
        $\sigma' \leftarrow \mathsf{Sign}(m', \mathsf{sk_{sig}}, \mathsf{pk_{san}}, \mathsf{tdpub}, \mathrm{ADM})$
      If $\sigma' \neq \perp$, set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m'\}$
      return $(m', \sigma')$
  return 1, if $a = b$
  return 0
```

Fig. 15. (Proof-Restricted) Transparency

```
**Experiment** $\mathsf{Sig\text{-}Acc}_{\mathcal{A}}^{\mathsf{3SSS}}(\lambda)$
  $\mathsf{pp_{3SSS}} \leftarrow \mathsf{PGen}(1^\lambda)$
  $(\mathsf{sk_{san}}, \mathsf{pk_{san}}) \leftarrow \mathsf{KGen_{san}}(\mathsf{pp_{3SSS}})$
  $(\mathsf{pk}^*, \mathsf{tdpub}^*, \pi^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{sk_{san}})}(\mathsf{pk_{san}})$
    for $i = 1, 2, \ldots, q$ let $(m'_i, \sigma'_i)$ and $(m_i, \mathrm{MOD}_i, \sigma_i, \mathsf{pk}_{\mathsf{sig},i}, \mathsf{tdpriv}_i)$
      index the answers/queries from/to $\mathsf{Sanit}$
  return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk_{san}}, \mathsf{tdpub}^*) = 1 \ \wedge$
    $\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^*, m^*, \mathsf{tdpub}^*) \neq (\mathsf{pk}_{\mathsf{sig},i}, m'_i, \mathsf{tdpub}) \ \wedge$
    $\mathsf{Judge}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk_{san}}, \pi^*) = \mathsf{San}$
  return 0
```

Fig. 16. Signer-Accountability

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, while ptd can always be derived from std by assumption.

Definition 19 (Sanitizer-Accountability). *An* 3SSS *is sanitizer-accountable, if for any ppt adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{San\text{-}Accountability}_{\mathcal{A}}^{\mathsf{3SSS}}(\lambda) = 1] \leq \nu(\lambda)$ *, where the experiment is defined in Fig. 17.*

```
$\textbf{Experiment}$ $\mathsf{San\text{-}Acc}_{\mathcal{A}}^{\mathsf{3SSS}}(\lambda)$
    $\mathsf{pp}_{\mathsf{3SSS}} \leftarrow \mathsf{PGen}(1^\lambda)$
    $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \mathsf{KGen}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{3SSS}})$
    $(\mathsf{pk}^*, \mathsf{tdpub}^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot,\cdot),\mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}},\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{sig}})$
        for $i = 1, 2, \ldots, q$ let $(m_i, \mathrm{ADM}_i, \mathsf{pk}_{\mathsf{san},i}, \mathsf{tdpub}_i)$ and $\sigma_i$
            index the queries/answers to/from $\mathsf{Sign}$
    $\pi \leftarrow \mathsf{Proof}(\mathsf{sk}_{\mathsf{sig}}, m^*, \sigma^*, \{(m_i, \sigma_i) \mid 0 < i \leq q\}, \mathsf{pk}^*, \mathsf{tdpub}^*)$
    return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*, \mathsf{tdpub}^*) = 1 \wedge$
        $\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^*, m^*, \mathsf{tdpub}^*) \neq (\mathsf{pk}_{\mathsf{san},i}, m_i, \mathsf{tdpub}_i) \wedge$
        $\mathsf{Judge}(m^*, \sigma^*, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}^*, \mathsf{tdpub}^*, \pi) = \mathsf{Sig}$
    return 0
```

Fig. 17. Sanitizer Accountability

Unlinkability. In line with Camenisch et al. [22], we do not consider unlinkability [19, 21, 36, 49] in our construction, as it seems to be very hard to achieve with the underlying construction paradigm.

Definition 20 (Secure 3SSS). *An 3SSS secure, if it is correct, private, transparent, unforgeable, immutable, immutable2, sanitizer-accountable and signer-accountable.*

Additional Building Blocks. For the construction, we require PRGs, PRFs and digital signatures (Σs). The formal definitions are given in App. A.

Construction. Subsequently, we present the construction of 3SSS. The underlying construction paradigm is an augmented version of the one by Brzuska et al. [17]. In a nutshell, each admissible block is hashed using a CHDLTT. Then, the hashes (along with the non-admissible blocks) are signed to achieve accountability. Note, however, that the used CHDLTT is not required to be unique.

Theorem 3. *If Σ and CHDLTT are secure, while PRF and PRG are both pseudo-random, then Construction 3 is secure.*

We sketch the proof below. The detailed proof of Theorem 3 is given in App. D.

Proof (Sketch). Unforgeability follows from the unforgeability of the underlying signature scheme and the public collision-resistance of the CHDLTT, as the adversary either needs to find a new signature or a collision in the hash. Immutability also follows from the unforgeability of the signature scheme, as each fixed block is signed. Immutability2, however, follows from private collision-resistance, as an adversary must find a collision for an honestly generated second trapdoor. Privacy and Transparency follow from the indistinguishability of the CHDLTT and pseudo-randomness of the PRF and PRG. Signer-accountability follows from the collision-resistance of the CHDLTT, as the proof must contain a collision which has never been published. Sanitizer-Accountability, follows from the unforgeability of the signature and the pseudo-randomness of PRF (and PRG).

Extensions. Subsequently, we present extensions to our basic scheme. We do not provide full-fledged formal security frameworks, as they are straightforward extensions to the one provided. However, these alterations increase the flexibility to delegate actions and may lead to new application scenarios for this primitive, and thus deserve to be mentioned.

$\mathsf{PGen}(1^\lambda)$**:** Let $\mathsf{pp}_\mathsf{chret} \leftarrow \mathsf{PGen}(1^\lambda)$. Return $\mathsf{pp}_\mathsf{3SSS} = \mathsf{pp}_\mathsf{chret}$.
$\mathsf{KGen}_\mathsf{sig}(\mathsf{pp}_\mathsf{3SSS})$**:** Let $(\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma) \leftarrow \mathsf{KGen}_\Sigma(1^\lambda)$. Pick a key for a PRF, i.e., $\kappa \leftarrow \mathsf{KGen}_\mathsf{prf}(1^\lambda)$. Return $((\kappa, \mathsf{sk}_\Sigma), \mathsf{pk}_\Sigma)$.
$\mathsf{KGen}_\mathsf{san}(\mathsf{pp}_\mathsf{3SSS})$**:** Let $(\mathsf{sk}_\mathsf{chret}, \mathsf{pk}_\mathsf{chret}) \leftarrow \mathsf{KGen}(\mathsf{pp}_\mathsf{chret})$. Return $(\mathsf{sk}_\mathsf{chret}, \mathsf{pk}_\mathsf{chret})$.
$\mathsf{TGen}(\mathsf{pp}_\mathsf{3SSS})$**:** Let $(\mathsf{std}, \mathsf{ptd}) \leftarrow \mathsf{TDGen}(\mathsf{pp}_\mathsf{chret})$. Return $(\mathsf{std}, \mathsf{ptd})$.

$\mathsf{Sign}(m, \mathsf{sk}_\mathsf{sig}, \mathsf{pk}_\mathsf{san}, \mathsf{tdpub}, \mathrm{ADM})$**:** If $\mathrm{ADM}(m) = 0$, return $\perp$. Draw $x \leftarrow \{0,1\}^\lambda$. Let $x' \leftarrow \mathsf{Eval}_\mathsf{prf}(\kappa, x)$. Let $\tau \leftarrow \mathsf{Eval}_\mathsf{prg}(x')$. For each block $i \in \mathrm{ADM}$, do $(h_i, r_i) \leftarrow \mathsf{Hash}(\mathsf{pk}_\mathsf{chret}, \mathsf{tdpub}, (i, m[i], \mathsf{pk}_\mathsf{sig}, \mathsf{tdpub}))$. For each block $i \notin \mathrm{ADM}$, let $h_i \leftarrow m[i]$, and $r \leftarrow \emptyset$. Let $(h_0, r_0) \leftarrow \mathsf{Hash}(\mathsf{pk}_\mathsf{chret}, (0, m, x, \ell, \tau, [h]_{1,\ell}, \mathsf{pk}_\mathsf{sig}, \mathsf{tdpub}))$. Sign $\sigma \leftarrow \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, ([h]_{0,\ell}, \ell, \mathsf{pk}_\mathsf{sig}, \mathsf{pk}_\mathsf{san}, \mathrm{ADM}, x, \mathsf{tdpub}))$. Return $(\sigma', ([h]_{0,\ell}, \mathrm{ADM}, \ell, x, \tau, [r]_{0,\ell}))$.
$\mathsf{Verify}(m, \sigma, \mathsf{pk}_\mathsf{sig}, \mathsf{pk}_\mathsf{san}, \mathsf{tdpub})$**:** For each block $i \in \mathrm{ADM}$, let $b_i \leftarrow \mathsf{Ver}(\mathsf{pk}_\mathsf{chret}, \mathsf{tdpub}, (i, m[i], \mathsf{pk}_\mathsf{sig}, \mathsf{tdpub}), r_i, h_i)$. Let $b_0 \leftarrow \mathsf{Ver}(\mathsf{pk}_\mathsf{chret}, \mathsf{tdpub}, (0, m, x, \ell, \tau, [h]_{1,\ell}, \mathsf{pk}_\mathsf{sig}, \mathsf{tdpub}), r_i, h_i)$. If any $b_i = 0$, return 0. Return $\mathsf{Verify}_\Sigma(\mathsf{pk}_\Sigma, ([h]_{0,\ell}, , \ell, \mathsf{pk}_\mathsf{sig}, \mathsf{pk}_\mathsf{san}, \mathrm{ADM}, x, \mathsf{tdpub}), \sigma')$.
$\mathsf{Sanit}(m, \mathrm{MOD}, \sigma, \mathsf{pk}_\mathsf{sig}, \mathsf{sk}_\mathsf{san}, \mathsf{tdpriv})$**:** Run $b \leftarrow \mathsf{Verify}(m, \sigma, \mathsf{pk}_\mathsf{sig}, \mathsf{pk}_\mathsf{san}, \mathsf{tdpub})$. If $b = 0$, return $\perp$. If $\mathrm{MOD}(\mathrm{ADM}) = 0$, return $\perp$. For each $(i, m[i]) \in \mathrm{MOD}$, let $r_i' \leftarrow \mathsf{Adap}(\mathsf{sk}_\mathsf{chret}, (i, m[i], \mathsf{pk}_\mathsf{sig}, \mathsf{tdpub}), (i, m'[i], \mathsf{pk}_\mathsf{sig}, \mathsf{tdpub}), r_i, h_i, \mathsf{std})$. Otherwise, let $r_i' \leftarrow r_i$. Draw $\tau' \leftarrow \{0,1\}^{2\lambda}$. Let $r_0' \leftarrow \mathsf{Adap}(\mathsf{sk}_\mathsf{chret}, (0, m, x, \ell, \tau, [h]_{1,\ell}, \mathsf{pk}_\mathsf{sig}, \mathsf{tdpub}), (0, m', x, \ell, \tau', [h]_{1,\ell}, \mathsf{pk}_\mathsf{sig}, \mathsf{std}), r_0, h_0, \mathsf{std})$. Return $(\sigma', ([h]_{0,\ell}, \mathrm{ADM}, \ell, x, \tau', [r']_{0,\ell}))$.

$\mathsf{Proof}(\mathsf{sk}_\mathsf{sig}, m, \sigma, \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}, \mathsf{pk}_\mathsf{san}, \mathsf{tdpub})$**:** Return $\perp$, if $0 = \mathsf{Verify}(m, \sigma, \mathsf{pk}_\mathsf{sig}, \mathsf{pk}_\mathsf{san}, \mathsf{tdpub})$. Verify each signature in the list, i.e., run $d_i \leftarrow \mathsf{Verify}(m_i, \sigma_i, \mathsf{pk}_\mathsf{sig}, \mathsf{pk}_\mathsf{san}, \mathsf{tdpub})$. If for any $d_i = 0$, return $\perp$. 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., $h_0 = h_0'$, where also $1 = \mathsf{Ver}(\mathsf{pk}_\mathsf{san}, (0, m, x, \ell, \tau, [h]_{1,\ell}, \mathsf{pk}_\mathsf{sig}, \mathsf{tdpub}), r_0, h_0)$, and $1 = \mathsf{Ver}(\mathsf{pk}_\mathsf{san}, (0, m', x, \ell, \tau', [h]_{1,\ell}, \mathsf{pk}_\mathsf{sig}, \mathsf{tdpub}), r_0', h_0')$ for some different tag τ' or message m'. Let this signature/message pair be $(\sigma', m') \in \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}$. Return $\pi = ((\sigma', m'), \mathsf{PRF}(\kappa, x))$, where x is contained in (σ, m).
$\mathsf{Judge}(m, \sigma, \mathsf{pk}_\mathsf{sig}, \mathsf{pk}_\mathsf{san}, \pi, \mathsf{tdpub})$**:** Check if π is of the form $((\sigma', m'), v)$ with $v \in \{0,1\}^\lambda$. If not, return Sig. Return $\perp$, if $0 = \mathsf{Verify}(m', \sigma', \mathsf{pk}_\mathsf{sig}, \mathsf{pk}_\mathsf{san}, \mathsf{tdpub})$, or $0 = \mathsf{Verify}(m, \sigma, \mathsf{pk}_\mathsf{sig}, \mathsf{pk}_\mathsf{san}, \mathsf{tdpub})$. Let $\tau'' \leftarrow \mathsf{PRG.Eval}_\mathsf{prg}(v)$. If $\tau' \neq \tau''$, return Sig, where τ' is taken from (σ', m'). If we have $h_0 = h_0'$, with a non-trivial colliding tuple $1 = \mathsf{Ver}(\mathsf{pk}_\mathsf{san}, (0, m, x, \ell, \tau, [h]_{0,\ell}, \mathsf{pk}_\mathsf{sig}, \mathsf{tdpub}), r_0, h_0) = \mathsf{Ver}(\mathsf{pk}_\mathsf{san}, (0, m', x, \ell, \tau', [h]_{0,\ell}, \mathsf{pk}_\mathsf{sig}, \mathsf{tdpub}), r_0', h_0')$, return San. Return Sig.

Construction 3: Secure 3SSS

Always Sanitizable Blocks. In the construction, the sanitizer can only sanitize, if it knows tdpriv. It may, however, be desirable that the sanitizer can always sanitize certain blocks, while others need tdpriv. This can easily be achieved by using a standard chameleon-hash instead of CHDLTT for those blocks. A formalization is straightforward and thus omitted.

Mixing Admissible Blocks. It may also be desirable that some blocks become sanitizable if tdpriv is released, and others if other tdprivs are released, i.e., there is more than one tdpriv for different blocks. Clearly, this can easily be achieved by using new tdpubs for each new block, perhaps even the same tdpubs for several blocks. Again, a formalization is straightforward and thus omitted.

We did not implement and measure any of these schemes, as the additional overhead is essentially one more single signature and some PRG and PRF evaluations, i.e., the lion's share is the evaluation of the chameleon-hash.

7 Additional Applications

This section is devoted to sketch some additional application scenarios of our new primitive. These use-cases include the already mentioned notion of group-level selectively revocable signatures and break-glass signatures.

Group-Level Selectively Revocable Signatures. Revoking (single) signatures is a long-standing open problem [8, 15, 43]. Using CHDLTT, however, we get one step closer to a satisfying solution. In particular, one can extend chameleon signatures [47] as follows: Instead of signing the hashed message m using the public key (of a standard chameleon-hash) of the recipient, one also uses ptd as an identifier for some group, e.g., an employee's name, or for a specific task. Then, to revoke signatures bound to this group, one simply releases std, i.e., makes it public, just as in a standard PKI where revoked public keys are made public. The corresponding ptd can be generated in advance by some entity, e.g., a server. All revoked signatures must then be considered invalid by a verifier, as all recipients could "fake" (i.e., adapt) arbitrary messages after revocation for that specific ptd. This is, essentially, the same argumentation as for standard chameleon signatures. Namely, as now any message can be "signed" by anyone[8] including the recipients, the signature loses all cryptographic value as an authentication of origin.

Break-The-Glass Emergency Signatures. Assume that some employees are not allowed to sign certain messages under "normal" conditions. Further assume the messages in question are an order form, or an approval, which requires explicit interaction with the department's head. If, however, the department's head is not reachable, e.g., due to sickness, but the signature must be generated, a designated employee may get access to some additional secret which allows to generate exactly this signature. If, however, this secret is revealed, it must be traceable that the secret was released, which can be enforced using standard policies. Our new primitive allows to technically map this workflow as follows: The employee able to sign a message in emergency generates a key pair for a standard signature scheme, but hashes an empty message to the corresponding ptd. The corresponding std can, e.g., be generated at some server within the same company. The employee can then gain access to std, and can thus sign any message it wants using Adap. This resembles a more sophisticated version of blank signatures [42]. This also resembles the emergency modes of access control, known as 'Break-Glass' or 'Break-The-Glass' (BTG), but for signature generation, e.g., those found in hospital information systems [35], and the server can log the distributed std to generate an audit trail of this emergency event.

8 Conclusion

We have introduced the notion of chameleon-hashes with dual long-term trapdoors. These are a generalization of chameleon-hashes with ephemeral trapdoors introduced by Camenisch et al. at PKC '17. They allow that the second trapdoor can be reused across multiple hashes. This primitive allows for even additional interesting application scenarios. These include three-party sanitizable signatures, group-level selectively revocable signatures and break-the-glass signatures. Our implementation's runtime measurements show that this primitive is practical.

References

1. Alsouri, S., Dagdelen, Ö., Katzenbeisser, S.: Group-based attestation: Enhancing privacy and management in remote attestation. In: Trust. pp. 63–77 (2010)
2. Ateniese, G., Chou, D.H., de Medeiros, B., Tsudik, G.: Sanitizable signatures. In: ESORICS. pp. 159–177 (2005)
3. Ateniese, G., Magri, B., Venturi, D., Andrade, E.R.: Redactable blockchain - or - rewriting history in bitcoin and friends. In: EuroS&P. pp. 111–126 (2017)
4. Ateniese, G., de Medeiros, B.: Identity-Based Chameleon Hash and Applications. In: Financial Cryptography. pp. 164–180 (2004)
5. Ateniese, G., de Medeiros, B.: On the key exposure problem in chameleon hashes. In: SCN (2004)
6. Bao, F., Deng, R.H., Ding, X., Lai, J., Zhao, Y.: Hierarchical identity-based chameleon hash and its applications. In: ACNS. pp. 201–219 (2011)
7. Beck, M.T., Camenisch, J., Derler, D., Krenn, S., Pöhls, H.C., Samelin, K., Slamanig, D.: Practical strongly invisible and strongly accountable sanitizable signatures. In: ACISP. pp. 437–452 (2017)
8. Beck, M.T., Krenn, S., Preiss, F., Samelin, K.: Practical signing-right revocation. In: Trust. pp. 21–39 (2016)

[8] Note, there might be corner cases for authorizing everyone as described by Pöhls [52].

9. 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)
10. Bellare, M., Ristov, T.: A characterization of chameleon hash functions and new, efficient designs. J. Cryptology 27(4), 799–823 (2014)
11. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: CCS. pp. 62–73. New York, NY, USA (1993)
12. Bilzhause, A., Huber, M., Pöhls, H.C., Samelin, K.: Cryptographically Enforced Four-Eyes Principle. In: ARES. pp. 760–767 (2016)
13. Bilzhause, A., Pöhls, H.C., Samelin, K.: Position paper: The past, present, and future of sanitizable and redactable signatures. In: Ares. pp. 87:1–87:9 (2017)
14. Blazy, O., Kakvi, S.A., Kiltz, E., Pan, J.: Tightly-secure signatures from chameleon hash functions. In: PKC. pp. 256–279 (2015)
15. Boneh, D., Ding, X., Tsudik, G., Wong, C.: A method for fast revocation of public key certificates and security capabilities. In: USENIX (2001)
16. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. 37(2), 156–189 (1988)
17. Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of Sanitizable Signatures Revisited. In: PKC (2009)
18. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Sanitizable signatures: How to partially delegate control for authenticated data. In: BIOSIG. pp. 117–128 (2009)
19. Brzuska, C., Fischlin, M., Lehmann, A., Schröder, D.: Unlinkability of Sanitizable Signatures. In: PKC. pp. 444–461 (2010)
20. Brzuska, C., Pöhls, H.C., Samelin, K.: Non-Interactive Public Accountability for Sanitizable Signatures. In: EuroPKI. pp. 178–193 (2012)
21. Brzuska, C., Pöhls, H.C., Samelin, K.: Efficient and Perfectly Unlinkable Sanitizable Signatures without Group Signatures. In: EuroPKI. pp. 12–30 (2013)
22. Camenisch, J., Derler, D., Krenn, S., Pöhls, H.C., Samelin, K., Slamanig, D.: Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures. In: PKC, Part II. pp. 152–182 (2017)
23. 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)
24. Canard, S., Jambert, A.: On extended sanitizable signature schemes. In: CT-RSA. pp. 179–194 (2010)
25. Canard, S., Jambert, A., Lescuyer, R.: Sanitizable signatures with several signers and sanitizers. In: AFRICACRYPT. pp. 35–52 (2012)
26. Chen, X., Tian, H., Zhang, F., Ding, Y.: Comments and improvements on key-exposure free chameleon hashing based on factoring. In: Inscrypt (2010)
27. Chen, X., Zhang, F., Kim, K.: Chameleon hashing without key exposure. In: ISC. pp. 87–98 (2004)
28. Chen, X., Zhang, F., Susilo, W., Mu, Y.: Efficient generic on-line/off-line signatures without key exposure. In: ACNS. pp. 18–30 (2007)
29. Chen, X., Zhang, F., Susilo, W., Tian, H., Li, J., Kim, K.: Identity-based chameleon hash scheme without key exposure. In: ACISP. pp. 200–215 (2010)
30. Damgård, I., Haagh, H., Orlandi, C.: Access control encryption: Enforcing information flow with cryptography. In: TCC-B. pp. 547–576 (2016)
31. 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)
32. Derler, D., Slamanig, D.: Rethinking privacy for extended sanitizable signatures and a black-box construction of strongly private schemes. In: ProvSec (2015)
33. Even, S., Goldreich, O., Micali, S.: On-line/off-line digital signatures. J. Cryptology 9(1), 35–67 (1996)
34. Fehr, V., Fischlin, M.: Sanitizable signcryption: Sanitization over encrypted data (full version). IACR Cryptology ePrint Archive, Report 2015/765 (2015)
35. Ferreira, A., Cruz-Correia, R., Antunes, L., Farinha, P., Oliveira-Palhares, E., Chadwick, D.W., Costa-Pereira, A.: How to break access control in a controlled manner. In: 19th IEEE Symposium on Computer-Based Medical Systems (CBMS'06). pp. 847–854 (2006)

36. 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)
37. 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)
38. Gao, W., Li, F., Wang, X.: Chameleon hash without key exposure based on schnorr signature. Computer Standards & Interfaces 31(2), 282–285 (2009)
39. Gao, W., Wang, X., Xie, D.: Chameleon Hashes Without Key Exposure Based on Factoring. J. Comput. Sci. Technol. 22(1), 109–113 (2007)
40. Goldwasser, S., Micali, S., Rivest, R.L.: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. SIAM Journal on Computing 17, 281–308 (1988)
41. Gong, J., Qian, H., Zhou, Y.: Fully-secure and practical sanitizable signatures. In: Inscrypt. vol. 6584, pp. 300–317 (2011)
42. Hanser, C., Slamanig, D.: Blank digital signatures. In: ASIACCS (2013)
43. Hanzlik, L., Kutylowski, M., Yung, M.: Hard invalidation of electronic signatures. In: ISPEC. pp. 421–436 (2015)
44. Hohenberger, S., Waters, B.: Short and stateless signatures from the RSA assumption. In: CRYPTO. pp. 654–670 (2009)
45. Höhne, F., Pöhls, H.C., Samelin, K.: Rechtsfolgen editierbarer signaturen. Datenschutz und Datensicherheit 36(7), 485–491 (2012)
46. Klonowski, M., Lauks, A.: Extended Sanitizable Signatures. In: ICISC (2006)
47. Krawczyk, H., Rabin, T.: Chameleon Hashing and Signatures. In: NDSS (2000)
48. Krenn, S., Samelin, K., Sommer, D.: Stronger security for sanitizable signatures. In: DPM. pp. 100–117 (2015)
49. 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)
50. 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)
51. Mohassel, P.: One-time signatures and chameleon hash functions. In: SAC (2010)
52. Pöhls, H.C.: Contingency revisited: Secure construction and legal implications of verifiably weak integrity. In: IFIPTM. IFIP AICT, vol. 401, pp. 136 – 150 (2013)
53. Pöhls, H.C., Peters, S., Samelin, K., Posegga, J., de Meer, H.: Malleable signatures for resource constrained platforms. In: WISTP. pp. 18–33 (2013)
54. Pöhls, H.C., Samelin, K.: Accountable redactable signatures. In: ARES (2015)
55. Pöhls, H.C., Samelin, K., Posegga, J.: Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In: ACNS. pp. 166–182 (2011)
56. Ren, Q., Mu, Y., Susilo, W.: Mitigating Phishing by a New ID-based Chameleon Hash without Key Exposure. In: AusCERT. pp. 1–13 (2007)
57. Shamir, A., Tauman, Y.: Improved online/offline signature schemes. In: CRYPTO. pp. 355–367 (2001)
58. Zhang, F., Safavi-naini, R., Susilo, W.: Id-based chameleon hashes from bilinear pairings. IACR Cryptology ePrint Archive 2003, 208 (2003)
59. Zhang, R.: Tweaking TBE/IBE to PKE transforms with Chameleon Hash Functions. In: ACNS. pp. 323–339 (2007)

A Security Definitions of Building Blocks

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

Pseudo-Random Functions 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 21 (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* $\left|\Pr[\mathsf{Pseudo\text{-}Randomness}_{\mathcal{A}}^{\mathsf{PRF}}(1^\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$*. The corresponding experiment is depicted in Fig. 18.*

Pseudo-Random Generators PRG.

Pseudo-Randomness. We require that PRG is actually pseudo-random.

```
Experiment $\mathsf{Pseudo\text{-}Randomness}_{\mathcal{A}}^{\mathsf{PRF}}(\lambda)$
  $\kappa \leftarrow \mathsf{KGen}_{\mathsf{prf}}(1^\lambda)$
  $b \leftarrow \{0,1\}$
  $f \leftarrow F_\lambda$
  $a \leftarrow \mathcal{A}^{\mathsf{Eval}'_{\mathsf{prf}}(\kappa,\cdot)}(1^\lambda)$
    where $\mathsf{Eval}'_{\mathsf{prf}}$ on input $\kappa, x$:
      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. 18. Pseudo-Randomness

```
Experiment $\mathsf{Pseudo\text{-}Randomness}_{\mathcal{A}}^{\mathsf{PRG}}(\lambda)$
  $b \leftarrow \{0,1\}$
  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. 19. Pseudo-Randomness

Definition 22 (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 $\left|\Pr[\mathsf{Pseudo\text{-}Randomness}_{\mathcal{A}}^{\mathsf{PRG}}(1^\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$. The corresponding experiment is depicted in Fig. 19.*

Digital Signatures Σ. Subsequently, we give the security properties required.

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 $\mathsf{Verify}_\Sigma(\mathsf{pk}_\Sigma, m, \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, m)) = 1$. This definition captures perfect correctness.

Unforgeability. Now, we define unforgeability of digital signature schemes, as given by Goldwasser et al. [40]. In a nutshell, we require that an adversary $\mathcal{A}$ cannot (except with negligible probability) come up with a valid signature σ^* for a new message m^*. Moreover, the adversary $\mathcal{A}$ can adaptively query for new signatures.

Definition 23 (Unforgeability). *A signature scheme Σ is unforgeable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν s.t. $\Pr[\mathsf{eUNF\text{-}CMA}_{\mathcal{A}}^{\Sigma}(1^\lambda) = 1] \leq \nu(\lambda)$. The corresponding experiment is depicted in Fig. 20.*

B Proof of Theorem 1

Proof. Correctness follows from inspection; the remaining properties are proven below.

Indistinguishability. This follows by a simple argument. In particular, consider the following sequence of games.

Game 0: The original indistinguishability game, where $b = 1$.
Game 1: As Game 0, but instead of calculating the hash h_1 as in the game, directly hash.

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

Fig. 20. Unforgeability

Transition - Game 0 → Game 1: Due to the indistinguishability of the chameleon hashes, this hop only changes the view of the adversary negligibly. Assume that the adversary can distinguish this hop. We can then construct an adversary $\mathcal{B}$ which breaks the indistinguishability of the chameleon hash. In particular, the reduction works as follows. $\mathcal{B}$ receives pk_c as it's own challenge, $\mathcal{B}$ embeds pk_c as $\mathsf{pk}_{\mathsf{chret}}$, and proceeds as in the prior hop, with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate h_1. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{ch\text{-}ind}}(\lambda)$ follows.

Game 2: As Game 1, but instead of calculating the hash h_2 as in the game, directly hash.

Transition - Game 1 → Game 2: Due to the indistinguishability of the chameleon hashes, this hop only changes the view of the adversary negligibly. Assume that the adversary can distinguish this hop. We can then construct an adversary $\mathcal{B}$ which breaks the indistinguishability of the chameleon hashes. In particular, the reduction works as follows. $\mathcal{B}$ receives pk'_c as it's own challenge, $\mathcal{B}$ embeds pk'_c as ptd, and proceeds as in the prior hop, with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate h_2. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{ch\text{-}ind}}(\lambda)$ follows.

We are now in the case $b = 0$. As each hop only changes the view of the adversary negligibly, this proves that our construction is indistinguishable.

Public Collision-Resistance. Let $\mathcal{A}$ be an adversary which breaks the public-collision resistance of our construction. We can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the collision-resistance of the underlying chameleon hash. We do so by a sequence of games:

Game 0: The original public collision-resistance game.

Game 1: As Game 0, but we abort if the adversary $\mathcal{A}$ outputs a forgery $(m^*, r^*, m'^*, r'^*, \mathsf{ptd}^*, h^*)$.

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. We can reduce the security to the collision-resistance of the underlying chameleon-hash. We can now construct an adversary $\mathcal{B}$ as follows. It receives pk'_c as the challenge public key. It uses this key to initialize $\mathcal{A}$. As the only oracle $\mathcal{B}$ has to simulate is the Adap' oracle, it proceeds as follows. On input $m, m', r, \mathsf{std}, \mathsf{ptd}, h$, $\mathcal{B}$ first checks, if the hash verifies. If not, it returns $\perp$. Otherwise, $\mathcal{B}$ computes $r'_2 \leftarrow \mathsf{CH.Adap}(\mathsf{std}, m, m', r_2, \mathsf{std}_2)$, where $m = (\mathsf{pk}'_c, \mathsf{ptd}, m)$, and $m' = (\mathsf{pk}'_c, \mathsf{ptd}, m')$. Adversary $\mathcal{B}$ then queries its own adaption oracle to receive r'_1 with the given m, m', and r_1, and gives (r'_1, r'_2) to $\mathcal{A}$. At some point, $\mathcal{A}$ returns $(m^*, r^*, m'^*, r'^*, h^*)$. Via assumption, we know that m^*, r^* w.r.t. m'^*, r'^* is "fresh", as $m \neq m'$ or $\mathsf{ptd}^* \neq \mathsf{ptd}'^*$, as ptd^* is part of the messages hashed, i.e., has never been returned by the Adap' oracle. Thus, $\mathcal{B}$ can return $(m^*, r^*_1, m'^*, r'^*_1, h^*_1)$ as its own forgery attempt. Now, the probability for our reduction to win is exactly $\Pr[E_1]$. That is, we have that $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{ch\text{-}coll}}(\lambda)$.

As the game hop only changes the view of the adversary negligibly, and the adversary has no way to forge a collision in Game 1, this proves that our construction is publicly collision-resistant.

Private Collision-Resistance. We use the following sequence of games to prove the private collision-resistance of our construction.

Game 0: The original private collision-resistance game.

Game 1: As Game 0, but the challenger aborts, if the adversary $\mathcal{A}$ outputs a valid forgery $(\mathsf{pk}^*, m^*, r^*, m'^*, r'^*, h^*)$.

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. We can now construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the collision-resistance of the underlying chameleon-hash. First, $\mathcal{B}$ receives $\mathsf{pk}_{\mathsf{ch}}^2$ from its own challenger. This is used as ptd to initialize $\mathcal{A}$. Then proceed as follows. For each query (with input $\mathsf{sk}_{\mathsf{ch}}^{1,i}$, and m), query the adaption oracle provided by the challenger. The collision w.r.t. $\mathsf{sk}_{\mathsf{ch}}^{1,i}$ can be generated honestly. At some point, $\mathcal{A}$ returns $(\mathsf{pk}^*, m^*, r^*, m'^*, r'^*, h^*)$. As we know that $h_2^* = h_2'^*$, and $(pk^*, \mathsf{ptd}, m'^*)$ must be fresh by assumption, $\mathcal{B}$ can return $((pk^*, \mathsf{ptd}, m^*), r_2^*, (pk^*, \mathsf{ptd}, m'^*), r_2'^*, h_2^*)$ as its own forgery attempt. Now, the probability for our reduction to win is exactly $\Pr[E_1]$. That is, we have that $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{ch\text{-}coll}}(\lambda)$.

As the game hop only changes the view of the adversary negligibly and the adversary has no other way to forge a collision, this proves that our construction is privately collision-resistant.

Uniqueness. We use the following sequence of games to prove that our construction is unique.

Game 0: The original uniqueness game.

Game 1: As Game 0, but the challenger aborts, if the adversary $\mathcal{A}$ outputs a valid forgery $(\mathsf{pk}^*, m^*, m'^*, r^*, r'^*, \mathsf{ptd}^*, h^*)$.

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. We can reduce the security to the uniqueness of the underlying chameleon-hash by constructing an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally. First, $\mathcal{B}$ receives $\mathsf{pp}_{\mathsf{chret}}$ from its own challenger. This is used to initialize $\mathcal{A}$. At some point, $\mathcal{A}$ returns $(\mathsf{pk}^*, m^*, m'^*, r^*, r'^*, \mathsf{ptd}^*, h^*)$. As we know that $r^* \neq r'^*$ if $\mathcal{A}$ wins, it follows that $r_1^* \neq r_1'^*$ or $r_2^* \neq r_2'^*$. In the first case, $\mathcal{B}$ can return $(\mathsf{pk}^*, m'^*, r_1^*, r_1'^*, h_1^*)$ as its own forgery attempt, where $m'^* = (\mathsf{pk}_{\mathsf{chret}}^*, \mathsf{ptd}^*, m^*)$. In the other case, it returns $(\mathsf{ptd}^*, m'^*, r_2^*, r_2'^*, h_2^*)$. Now, the probability for our reduction to win is exactly $\Pr[E_1]$. That is, we have that $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{ch\text{-}uniqueness}}(\lambda)$.

Thus proves, that our construction is unique.

C Proof of Theorem 2

Proof. We need to prove that our construction is indistinguishable, publicly collision-resistant, and privately collision-resistant.

Indistinguishability. It is easy to see that the above construction is indistinguishable; all values are chosen uniformly at random, and RSA defines a permutation with the restrictions given.

Public Collision-Resistance. We now prove that the above construction is collision-resistant.

Game 0: This is the original public collision-resistance game.

Game 1: As Game 0, but instead of using the e from the system parameters (here, $e > n^3$), we embed the e received from a one-more RSA challenger as $\mathsf{pp_{chret}}$. Note that we can still determine d—and therefore honestly simulate all oracles—as we do not use n from the challenger at this point and can thus freely choose it.

Transition - Game 0 → Game 1: This does not change the view of the adversary, as the received e is distributed identically to the real game. $|\Pr[S_0] - \Pr[S_1]| = 0$ follows.

Game 2: As Game 1, but we now abort, if the adversary was able to generate a collision $(m^*, r^*, m'^*, r'^*, \mathsf{ptd}^*, h^*)$. Let this event be denoted E_2.

Transition - Game 1 → Game 2: Assume that event E_2 does happen with non-negligible probability. We can then build an adversary $\mathcal{B}$ which breaks the one-more RSA-inversion assumption. Without loss of generality, we assume that the adversary makes all the random oracle queries before outputting the messages (otherwise, $\mathcal{B}$ does them). The adversary $\mathcal{B}$ proceeds as follows. In the first step, the challenge n_c is embedded in $\mathsf{pk_{chret}}$ as n. Clearly, as the distributions are the same, this is only a conceptual change so far. In the second step, for each *new* random-oracle query m_i with format (m, n, n') to $\mathcal{H}_{nn'}$, $\mathcal{B}$ asks its challenge oracle $\mathcal{C}$ to provide a challenge $c_i \in \mathbb{Z}^*_n$. However, it may happen that $c_i \notin \mathbb{Z}^*_{nn'}$ (note, we have a family of random-oracles!). In this case, request a new c_i from the challenge oracle till the condition holds.[9] Draw $u_i \leftarrow \mathbb{Z}^*_{nn'}$ and record $(m_i, n', c_i, u_i, \perp)$. Embed $c_i u_i^e \bmod nn'$ as the random-oracle response for m_i. Note, this value is distributed perfectly uniformly in $\mathbb{Z}^*_{nn'}$. However, it remains open how to simulate the adaption oracle. Assume, for now, that m_0 is supposed to be adapted to m_1, while the second modulus is n'. If $m_0 = m_1$, or $n = n'$, proceed as in the algorithm. If there is no tuple (m_0, n', c_0, u_0, z_0), with $z_0 \neq \perp$, query the inversion oracle with c_0 to receive z_0. Update record $(m_0, n', c_0, u_0, \perp)$ to (m_0, n', c_0, u_0, z_0). If there is no tuple (m_1, n', c_1, u_1, z_1), with $z_1 \neq \perp$, query the inversion oracle with c_1 to receive z_1. Update record $(m_1, n', c_1, u_1, \perp)$ to (m_1, n', c_1, u_1, z_1). Calculate the collision mod$n$ as $rz_0(z_1)^{-1}$. The collision modn' can be calculated honestly, as the factors are known. Combine the result modnn' using the Chinese remainder theorem, and return it. Eventually, the adversary returns $(m^*, r^*, m'^*, r'^*, \mathsf{ptd}^*, h^*)$. We then know (by construction) that $\mathcal{H}_{nn'}(m^*)r^{*e} \equiv \mathcal{H}_{nn'}(m'^*)r'^{*e} \bmod nn'$. If there is no record for m^* *and* no record for m'^*, query the inversion for oracle for the root z^* for the c^* embedded in $\mathcal{H}_{nn'}(m^*)$, and update record $(m^*, n', c^*, r^*, \perp)$ to (m^*, n', c^*, u^*, z^*). Then, by definition, we know that $m'^* = (m''^*, n, \mathsf{ptd}^*)$ is fresh (with $\mathsf{ptd}^* = n'$), and there exists a record $(m'^*, n', c'^*, u'^*, \perp)$. We can then extract $\mathcal{H}_{nn'}(m'^*)^d$ by calculating $\mathcal{H}_{nn'}(m'^*)^d \equiv r'^{*-1}z^*r^* \bmod nn'$, and extract the root of c'^* by multiplying it with $u'^{*-1} \bmod nn'$, resulting in z'^*. As therefore the adversary $\mathcal{A}$ has inverted more challenges than the inversion oracle was queried, $\mathcal{B}$ can return the list $\{(c_i, z_i)\}$ for each entry where $(m_i, n'_i, c_i, r_i, z_i)$, $z_i \neq \perp$ exists, along with $(c'^*, z'^* \bmod n)$. Note, the probability for our reduction to win is exactly $\Pr[E_2]$. That is, we have that $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{om\text{-}rsa}}(\lambda)$ follows.

As now the adversary has no other way to win its game, public collision-resistance is proven, as each hop only changes the view of the adversary negligibly.

Private Collision-Resistance. We now prove that the above construction is collision-resistant.

Game 0: This is the original private collision-resistance game.

Game 1: As Game 0, but instead of using the e from the system parameters (here, $e > n^3$), we embed the e received from a RSA challenger as $\mathsf{pp_{chret}}$. Note that we can still determine d—and therefore honestly simulate all oracles—as we do not use n' from the challenger at this point and can thus freely choose it.

Transition - Game 0 → Game 1: This does not change the view of the adversary, as the received e is distributed identically to the real game. $|\Pr[S_0] - \Pr[S_1]| = 0$ follows.

[9] Camenisch et al. [22] show that this directly provides a polynomial bound on the expected number of needed samples.

Game 2: As Game 1, but we now abort, if there are random-oracle collisions in any random oracle. Let this event be E_2.

Transition - Game 1 → Game 2: Event E_2 cannot happen with non-negligible probability due to the birthday-bound. $|\Pr[S_1] - \Pr[S_2]| \leq \frac{q_h^2}{2^\lambda}$ follows, where q_h is the number of random-oracle queries.

Game 3: We now abort, if the adversary was able to output a tuple $(\mathsf{pk}^*, m^*, r^*, m'^*, r'^*, h^*)$ which breaks the private collision-resistance of our construction. Let this event be E_3.

Transition - Game 2 → Game 3: Assume that event E_3 does happen with non-negligible probability. We can then build an adversary $\mathcal{B}$ which breaks the one-more RSA-inversion assumption. Without loss of generality, we assume that the adversary makes all the random oracle queries before outputting the messages (otherwise, $\mathcal{B}$ does them). The adversary $\mathcal{B}$ proceeds as follows. In the first step, the challenge n_c is embedded in ptd as n'. Clearly, as the distributions are the same, this is only a conceptual change so far. In the second step, for each *new* random-oracle query m_i with format (m, n, n') to $\mathcal{H}_{nn'}$, $\mathcal{B}$ asks its challenge oracle $\mathcal{C}$ to provide a challenge $c_i \in \mathbb{Z}_n^*$. However, it may happen that $c_i \notin \mathbb{Z}_{nn'}^*$. In this case, request a new c_i from the challenge oracle till the condition holds. As before, this only implies a polynomial loss. Draw $u_i \leftarrow \mathbb{Z}_{nn'}^*$ and record $(m_i, n, c_i, u_i, \perp)$. Embed $c_i u_i^e \bmod nn'$ as the random-oracle response for m_i. Note, this value is distributed perfectly uniformly in $\mathbb{Z}_{nn'}^*$. However, it remains open how to simulate the adaption oracle. Assume, for now, that m_0 is supposed to be adapted to m_1, while the second modulus is n (defined by $\mathsf{pk}_{\mathsf{ch}}$). If $m_0 = m_1$, or $n = n'$, proceed as in the algorithm. If there is no tuple (m_0, n, c_0, u_0, z_0), with $z_0 \neq \perp$, query the inversion oracle with c_0 to receive z_0. Update record $(m_0, n, c_0, u_0, \perp)$ to (m_0, n, c_0, u_0, z_0). If there is no tuple (m_1, n, c_1, u_1, z_1), with $z_1 \neq \perp$, query the inversion oracle with c_1 to receive z_1. Update record $(m_1, n, c_1, u_1, \perp)$ to (m_1, n, c_1, u_1, z_1). Calculate the collision mod n' as $rz_0(z_1)^{-1}$. The collision mod n can be calculated honestly, as the factors are known. Combine the result mod nn' using the Chinese remainder theorem, and return it. Eventually, the adversary returns $(\mathsf{pk}^*, m^*, r^*, m'^*, r'^*, h^*)$. We then know (by construction) that $\mathcal{H}_{nn'}(m^*)r^{*e} \equiv \mathcal{H}_{nn'}(m'^*)r'^{*e} \bmod nn'$. If there is no record for m^* *and* no record for m'^*, query the inversion for oracle for the root z^* for the c^* embedded in $\mathcal{H}_{nn'}(m^*)$, and update record $(m^*, n, c^*, r^*, \perp)$ to (m^*, n, c^*, u^*, z^*). Then, by definition, we know that $m'^* = (m''^*, \mathsf{pk}^*, n')$ is fresh (with $\mathsf{pk}^* = n$), and there exists a record $(m'^*, n, c'^*, u'^*, \perp)$. We can then extract $\mathcal{H}_{nn'}(m'^*)^d$ by calculating $\mathcal{H}_{nn'}(m'^*)^d \equiv r'^{*-1} z^* r^* \bmod nn'$, and extract the root of c'^* by multiplying it with $u'^{*-1} \bmod nn'$, resulting in z'^*. As therefore the adversary $\mathcal{A}$ has inverted more challenges than the inversion oracle was queried, $\mathcal{B}$ can return the list $\{(c_i, z_i)\}$ for each entry where $(m_i, n_i, c_i, r_i, z_i)$, $z_i \neq \perp$ exists, along with $(c'^*, z'^* \bmod n)$. Now, the probability for our reduction to win is exactly $\Pr[E_3]$. That is, we have that $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{om\text{-}rsa}}(\lambda)$ follows. Thus, private collision-resistance is proven.

As now the adversary has no other way to win its game, private collision-resistance is proven, as each hop only changes the view of the adversary negligibly.

Uniqueness. We use the following sequence of games to prove that our construction is unique.

Game 0: The original uniqueness game.

Game 1: As Game 0, but the challenger aborts, if the adversary $\mathcal{A}$ outputs a valid forgery $(\mathsf{pk}^*, m^*, m'^*, r^*, r'^*, \mathsf{ptd}^*, h^*)$.

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. This cannot happen, as RSA, with the given restrictions, forms a permutation while random oracle behaves like functions.

Thus this proves, that our construction is unique.

D Proof of Theorem 3

Now we prove the security of Construction 3.

Proof. Correctness follows by inspection. Each property is proven 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 $(m^*, \sigma^*, \mathsf{tdpub}^*)$ with $\sigma^* = (\sigma'^*, ([h^*]_{1,\ell^*}, \mathrm{ADM}^*, \ell^*, x^*, \tau^*, [r^*]_{0,\ell^*}))$, where $m'^* = ([h^*]_{1,\ell^*}, \ell^*, \mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}, \mathrm{ADM}^*, x^*, \mathsf{tdpub}^*)$ was never signed by the signing oracle. Let this event be E_1.

Transition - Game 0 → Game 1: Clearly, if $([h^*]_{1,\ell^*}, \ell^*, \mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}, \mathrm{ADM}^*, x^*, \mathsf{tdpub}^*)$ was never signed by the challenger, this tuple breaks the unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a unforgeability challenger and embed it as $\mathsf{pk}_{\mathsf{sig}}$. For every required "inner" signature σ', 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 $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{eunf\text{-}cma}}(\lambda)$.

Game 2: Among others, we now have established that the adversary can no longer win by modifying $\mathsf{pk}_{\mathsf{sig}}$, $\mathsf{pk}_{\mathsf{san}}$, and tdpub. We proceed as in Game 1, but abort if the adversary outputs a forgery $(m^*, \sigma^*, \mathsf{tdpub}^*)$, 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 the public 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 with the same tdpub. Now, assume that we obtain pk_c from a public-collision freeness challenger. If E_2 happens, there must be a previous oracle query with associated values $(0, m^*, x^*, \ell^*, \tau^*, [h^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{tdpub}^*)$ 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 $(0, m^*, x^*, \ell^*, \tau^*, [h^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{tdpub}^*) \neq (0, m'^*, x'^*, \ell'^*, \tau'^*, [h'^*]_{1,\ell'^*}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{tdpub}^*)$, and can thus output $((0, m^*, x^*, \ell^*, \tau^*, [h^*]_{1,\ell^*}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{tdpub}^*), r^*_0, (0, m'^*, x'^*, \ell'^*, \tau'^*, [h'^*]_{1,\ell'^*}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{tdpub}^*), r'^*_0, h^*_0, \mathsf{tdpub})$ 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. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{chret\text{-}pub\text{-}coll\text{-}res}}(\lambda)$ follows.

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 $(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{tdpub}^*)$ with $\sigma^* = (\sigma'^*, ([h^*]_{1,\ell^*}, \mathrm{ADM}^*, \ell^*, x^*, \tau^*, [r^*]_{0,\ell^*}))$ where $m'^* = ([h^*]_{1,\ell^*}, \ell^*, \mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}, \mathrm{ADM}^*, x^*, \mathsf{tdpub}^*)$ 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 $([h^*]_{1,\ell^*}, \mathrm{ADM}^*, \ell^*, x^*, \tau^*, [r^*]_{0,\ell^*})$ was never signed by the challenger, this tuple breaks the unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a unforgeability challenger and embed it as $\mathsf{pk}_{\mathsf{sig}}$. For every required "inner" signature σ', 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 $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{eunf\text{-}cma}}(\lambda)$.

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 $\mathsf{pk_{sig}}$, ADM, tdpub, and $\mathsf{pk_{san}}$, 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. 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 $\sigma''^* = (\sigma'''^*, ([h'^*]_{0,\ell'^*}, \mathrm{ADM}'^*, \ell'^*, x'^*, \tau'^*, [r'^*]_{0,\ell'^*}))$ 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 $\mathsf{Ver}(\mathsf{pk_{ch}}, (i, m^*[i], \mathsf{pk_{sig}}, \mathsf{tdpub}^*), r_i^*, h_i^*) = \mathsf{Ver}(\mathsf{pk_{ch}}, (i, m'^*[i], \mathsf{pk_{sig}}, \mathsf{tdpub}^*), r_i'^*, h_i^*) = 1$, where $m^*[i] \neq m'^*[i]$ by assumption, but for which ADM does not match. However, tampering with ADM was already ruled out. $|\Pr[S_1] - \Pr[S_2]| = 0$ follows.

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.

Immutability2. We prove immutability2 using a sequence of games.

Game 0: The immutability2 game.

Game 1: As Game 0, but we abort if the adversary was able to output $(m^*, \sigma^*, \mathsf{pk^*_{sig}}, \mathsf{pk^*_{san}}, \pi^*)$ with the giving winning conditions as defined in Fig. 13. Let this event be E_1.

Transition - Game 0 → Game 1: Assume that E_1 is non-negligible. We can then construct a reduction which breaks the private collision-resistance of CHDLTT. In particular, the reduction proceeds as follows. By construction, we know that $(0, m^*, x^*, \ell^*, \tau^*, [h^*]_{1,\ell^*}, \mathsf{pk^*_{sig}}, \mathsf{tdpub}) \neq (0, m'^*, x'^*, \ell'^*, \tau'^*, [h'^*]_{1,\ell'^*}, \mathsf{pk^*_{sig}}, \mathsf{tdpub})$, contained in π, while both have the same hash h_0^*. Moreover, we know that either m^* or $\mathsf{pk^*_{sig}}$ is fresh by definition. The reduction now works as follows. It receives tdpub_c from its own challenger, and embeds it as tdpub. Then, as clarified before, we can return $(\mathsf{pk^*_{san}}, (0, m^*, x^*, \ell^*, \tau^*, [h^*]_{1,\ell^*}, \mathsf{pk^*_{sig}}, \mathsf{tdpub}), r_0^*, (0, m'^*, x'^*, \ell'^*, \tau'^*, [h'^*]_{1,\ell'^*}, \mathsf{pk^*_{sig}}, \mathsf{tdpub}), r_0'^*, h_0^*)$ as its own forgery. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{chret\text{-}priv\text{-}coll\text{-}res}}(\lambda)$ follows.

As each hop changes the view of the adversary only negligibly, immutability2 is proven.

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^*) for which m^* 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. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{3sss\text{-}unf}}(\lambda)$ follows.

Game 2: As Game 1, but we abort if the adversary outputs a forgery $(m^*, \sigma^*, \mathsf{tdpub}^*)$ with $\sigma^* = (\sigma'^*, ([h^*]_{0,\ell^*}, \mathrm{ADM}^*, \ell^*, x^*, \tau^*, [r^*]_{0,\ell^*}))$, where $m'^* = ([h^*]_{0,\ell^*}, \ell^*, \mathsf{pk^*_{sig}}, \mathsf{pk^*_{san}}, \mathrm{ADM}^*, x^*, \mathsf{tdpub}^*)$ was never signed by the signing oracle. Let this event be E_2.

Transition - Game 1 → Game 2: Clearly, if $([h^*]_{0,\ell^*}, \ell^*, \mathsf{pk^*_{sig}}, \mathsf{pk^*_{san}}, \mathrm{ADM}^*, x^*, \mathsf{tdpub}^*)$ was never signed by the challenger, this tuple breaks the unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a unforgeability challenger and embed it as $\mathsf{pk_{sig}}$. For every required "inner" signature σ', 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 $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{eunf\text{-}cma}}(\lambda)$.

Game 3: As Game 2, but instead of hashing the blocks $(i, m_b^*[i], \mathsf{pk_{sig}}, \mathsf{tdpub})$ for the inner chameleon-hashes using Hash, and then Adap to $(i, m[i]^*, \mathsf{pk_{sig}}, \mathsf{tdpub})$, we directly apply Hash to $(i, m[i], \mathsf{pk_{sig}}, \mathsf{tdpub})$.

Transition - Game 2 → Game 3: 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 $\mathsf{pk_{ch}}$, and ptd_c as tdpub, where $m[i]$ is admissible. It proceeds honestly with the exception that it uses the HashOrAdapt oracle to generate that inner hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{chret\text{-}ind}}(\lambda)$ follows.

Game 4: As Game 3, but instead of adapting $(0, m, x, \ell, \tau, [h]_{0,\ell}, \mathsf{pk_{sig}}, \mathsf{tdpub})$ to the new values, we directly use Hash.

Transition - Game 3 → Game 4: 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 $\mathsf{pk_{ch}}$ (and ptd_c as tdpub), and proceeds honestly with the exception that it uses the HashOrAdapt oracle to generate the outer hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{chret\text{-}ind}}(\lambda)$ 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 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^*) for which m^* was never returned by any of the calls to the sanitization, or signature generation, oracles. 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 $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{3sss\text{-}unf}}(\lambda)$.

Game 2: As Game 1, but we abort if the adversary outputs a forgery $(m^*, \sigma^*, \mathsf{tdpub}^*)$ with $\sigma^* = (\sigma'^*, ([h^*]_{0,\ell^*}, \mathrm{ADM}^*, \ell^*, x^*, \tau^*, [r^*]_{0,\ell^*}))$, where $m'^* = ([h^*]_{0,\ell^*}, \ell^*, \mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}, \mathrm{ADM}^*, x^*, \mathsf{tdpub}^*)$ was never signed by the signing oracle. Let this event be E_2.

Transition - Game 1 → Game 2: Clearly, if $([h^*]_{0,\ell^*}, \ell^*, \mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}, \mathrm{ADM}^*, x^*, \mathsf{tdpub}^*)$ was never signed by the challenger, this tuple breaks the unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a unforgeability challenger and embed it as $\mathsf{pk_{sig}}$. For every required "inner" signature σ', 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 $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{eunf\text{-}cma}}(\lambda)$.

Game 3: As Game 2, but instead of computing $x'_0 \leftarrow \mathsf{Eval_{prf}}(\kappa, x_0)$, we set $x'_0 \leftarrow \{0,1\}^\lambda$ within every call to Sign in the Sanit/Sign oracle.

Transition - Game 2 → Game 3: A distinguisher between these two games straightforwardly yields a distinguisher for the PRF. Thus, we have $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{ind\text{-}prf}}(\lambda)$.

Game 4: As Game 3, but instead of computing $\tau \leftarrow \mathsf{Eval_{prg}}(x'_0)$, we set $\tau \leftarrow \{0,1\}^{2\lambda}$ for every call to Sign within the Sanit/Sign oracle.

Transition - Game 3 → Game 4: A distinguisher between these two games yields a distinguisher for the PRG using a hybrid argument. Thus, we have $|\Pr[S_3] - \Pr[S_4]| \leq q_s \nu_{\mathsf{ind\text{-}prg}}(\lambda)$, where q_s is the number of calls to the PRG.

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

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 $\mathsf{pk}_{\mathsf{ch}}$, and ptd_c as tdpub. 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 $\mathcal{B}$. $|\Pr[S_4] - \Pr[S_5]| \leq \nu_{\mathsf{chret\text{-}ind}}(\lambda)$ follows.

Game 6: As Game 5, but instead of hashing and then adapting the outer hash, we directly hash the message, i.e., $(0, m, x, \ell, \tau, [h]_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{tdpub})$.

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. In particular, the reduction works as follows. $\mathcal{B}$ receives pk_c as it's own challenge, embeds pk_c as $\mathsf{pk}_{\mathsf{ch}}$ (as well as ptd_c as tdpub), 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 $\mathcal{B}$. $|\Pr[S_5] - \Pr[S_6]| \leq \nu_{\mathsf{chret\text{-}ind}}(\lambda)$ follows.

We are in the case $b = 1$. As each hop only changes the view negligibly, transparency is proven.

Signer-Accountability. We prove that our construction is 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 sanitization oracle draws a tag τ' which is in the range of the PRG. Let this event be E_1.

Transition - Game 0 → Game 1: This hop is indistinguishable by a standard statistical argument: at most 2^λ values lie in the range of the PRG. $|\Pr[S_0] - \Pr[S_1]| \leq \frac{q_s 2^\lambda}{2^{2\lambda}} = \frac{q_s}{2^\lambda}$ follows, where q_s is the number of sanitizing requests. Note, this also means, that there exists no valid pre-image x.

Game 2: As Game 1, but we abort, if a tag in the sanitization oracle was drawn twice. Let this event be E_2.

Transition - Game 1 → Game 2: This hop is indistinguishable due the birthday paradox. $|\Pr[S_1] - \Pr[S_2]| \leq \frac{q_s^2 2}{2^\lambda}$ follows, where q_s is the number of sanitizing requests.

Game 3: As Game 2, but we now abort, if the adversary was able to find $(\mathsf{pk}^*, \mathsf{tdpub}^*, \pi^*, m^*, \sigma^*)$ for some $(0, m^*, x^*, \ell^*, \tau^*, [h^*]_{1,\ell^*}, \mathsf{pk}^*, \mathsf{tdpub}^*)$ in (m^*, σ^*), which was never returned by the sanitization oracle. Let this event be E_3.

Transition - Game 2 → Game 3: 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_2 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_3 happens we can output $((0, m^*, x^*, \ell^*, \tau^*, [h^*]_{1,\ell^*}, \mathsf{pk}^*, \mathsf{tdpub}^*), r_0^*, (0, m'^*, x'^*, \ell'^*, \tau'^*, [h'^*]_{1,\ell^*}, \mathsf{pk}^*, \mathsf{tdpub}^*), r_0'^*, h_0^*)$, 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. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{chret\text{-}pub\text{-}coll\text{-}res}}(\lambda)$ follows.

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 $(\mathsf{pk}^*, \mathsf{tdpub}^*, m^*, \sigma^*)$ with $\sigma^* = (\sigma'^*, ([h^*]_{1,\ell^*}, \mathrm{ADM}^*, \ell^*, x^*, \tau^*, [r^*]_{0,\ell^*}))$, where $m'^* = ([h^*]_{1,\ell^*}, \ell^*, \mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}, \mathrm{ADM}^*, x^*, \mathsf{tdpub}^*)$ was never signed by the signing oracle.

Transition - Game 0 $\rightarrow$ Game 1: Let us use E_1 to refer to the abort event. Clearly, if $([h^*]_{1,\ell^*}, \ell^*, \mathsf{pk}^*_{\mathsf{sig}}, \mathsf{pk}^*_{\mathsf{san}}, \mathrm{ADM}^*, x^*, \mathsf{tdpub}^*)$ was never obtained from the challenger, this σ'^* breaks the 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', 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 $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{eunf\text{-}cma}}(\lambda)$.

In Game 1 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 $h_0^* = h_0'^*$. In this case, the Judge algorithm returns San, and $\Pr[S_1] = 0$ follows, which concludes the proof.