This is the full version of a paper which appears in Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings. LNCS 10175. pages 152–182. © IACR, 2017.

Chameleon-Hashes with Ephemeral Trapdoors

And Applications to Invisible Sanitizable Signatures

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

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

Abstract. A chameleon-hash function is a hash function that involves a trapdoor the knowledge of which allows one to find arbitrary collisions in the domain of the function. In this paper, we introduce the notion of *chameleon-hash functions with ephemeral trapdoors*. Such hash functions feature additional, i.e., ephemeral, trapdoors which are chosen by the party computing a hash value. The holder of the main trapdoor is then unable to find a second pre-image of a hash value unless also provided with the ephemeral trapdoor used to compute the hash value. We present a formal security model for this new primitive as well as provably secure instantiations. The first instantiation is a generic black-box construction from any secure chameleon-hash function. We further provide three direct constructions based on standard assumptions. Our new primitive has some appealing use-cases, including a solution to the long-standing open problem of *invisible* sanitizable signatures, which we also present.

1 Introduction

Chameleon-hash functions, also called trapdoor-hash functions, are hash functions that feature a trapdoor that allows one to find arbitrary collisions in the domain of the functions. However, chameleon-hash functions are collision resistant as long as the corresponding trapdoor (or secret key) is not known. More precisely, a party who is privy of the trapdoor is able to find arbitrary collisions in the domain of the function. Example instantiations include trapdoor-commitment, and equivocal commitment schemes.

One prominent application of this primitive are chameleon signatures [KR00]. Here, the intended recipient—who knows the trapdoor—of a signature σ for

[‡] Supported by EU ERC PERCY, grant agreement n°32131.

[‖] Supported by EU H2020 project PRISMACLOUD, grant agreement n°644962.

a message m can equivocate it to another message m' of his choice. This, in turn, means that a signature σ cannot be used to convince any other party of the authenticity of m, as the intended recipient could have "signed" arbitrary messages on its own. Many other applications appear in the literature, some of which we discuss in the related work section. However, all current constructions are "all-or-nothing" in that a party who computes a hash with respect to some public key cannot prevent the trapdoor holder from finding collisions. This can be too limiting for some use-cases.

Contribution. We introduce a new primitive dubbed chameleon-hash functions with ephemeral trapdoors. In a nutshell, this primitive requires that a collision in the hash function can be computed only when two secrets are known, i.e., the main trapdoor, and an ephemeral one. The main trapdoor is the secret key corresponding to the chameleon-hash function public key, while the second, ephemeral, trapdoor is generated by the party computing the hash value. The latter party can then decide whether the holder of the long-term secret key shall be able to equivocate the hash by providing or withholding the second trapdoor information. We present a formal security model for this new primitive. Furthermore, we present stronger definitions for existing chameleon-hash functions not considered before, including the new notion of uniqueness, and show how to construct chameleon-hash functions being secure in this stronger model. These new notions may also be useful in other scenarios.

Additionally, we provide four provably secure constructions for chameleon-hash functions with ephemeral trapdoors. The first is bootstrapped, while the three direct constructions are built on RSA-like and the DL assumption. Our new primitive has some interesting applications, including the first provably secure instantiation of *invisible* sanitizable signatures, which we also present. Additional applications of our new primitive may include revocable signatures [HKY15], but also simulatable equivocable commitments [Fis01]. However, in contrast to equivocable commitments, we want that parties can actually equivocate, not only a simulator. Therefore, we chose to call this primitive a chameleon-hash function rather than a commitment. Note, the primitive is different from "double-trapdoor chameleon-hash functions" [BCG07, CRFG08, LZCS16], where knowing one out of two secrets is enough to produce collisions.

Related Work and State-of-the-Art. Chameleon-hash functions were introduced by Krawczyk and Rabin [KR00], and are based on some first ideas given by Brassard et al. [BCC88]. Later, they have been ported to the identity-based setting (ID-based chameleon-hash functions), where the holder of a master secret key can extract new secret keys for each identity [AdM04a, BDD$^+$11, CTZD10, RMS07, ZSnS03]. These were mainly used to tackle the key-exposure problem [AdM04b, KR00]. Key exposure means that seeing a single collision in the hash allows to find further collisions by extracting the corresponding trapdoor. This problem was then directly solved by the introduction of "key-exposure free" chameleon-hash functions [AdM04b, GLW09, GWX07, RMS07], which prohibit extracting the (master) secret key. This allows for the partial re-use of generated key material. Brzuska et al. then proposed a formal framework for tag-based chameleon-hashes secure under random-tagging attacks, i.e., random identities [BFF+09].

Beside this "plain" usage of the aforementioned primitive, chameleon-hash functions also proved useful in other areas such as on/offline signatures [CZSM07, EGM96, ST01], (tightly) secure signature schemes [BKKP15, HW09, Moh10], but also sanitizable signature schemes [ACdMT05, BFF+09, GQZ11] and identity-based encryption schemes [Zha07]. Moreover they are useful in context of trapdoor-commitments, direct anonymous attestation, Σ-protocols, and distributed hashing [ADK10, BR14, BCC88, Fis01].

Additional related work is discussed when presenting the application of our new primitive.

2 Preliminaries

Let us give our notation, the required assumptions, building blocks, and the extended framework for chameleon-hashes (without ephemeral trapdoors) first.

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 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. For the remainder of this paper, all algorithms are ppt if not explicitly mentioned otherwise. Most algorithms may return a special error symbol $\perp \notin \{0,1\}^*$, denoting an exception. If S is a set, we write $a \leftarrow S$ to denote that a is chosen uniformly at random from S. For a message $m = (m[1], m[2], \ldots, m[\ell])$, we call $m[i]$ a block, while $\ell \in \mathbb{N}$ denotes the number of blocks in a message m. For a list we require that we have 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 $\nu(n) \leq n^{-k}$, $\forall n > n_0$. For certain security properties we require that values only have one canonical representation, e.g., a "4" is not the same as a "04", even if written as elements of $\mathbb{N}$ for brevity. Finally, for a group G we use G^* to denote $G \setminus \{1_G\}$.

2.1 Assumptions

Discrete Logarithm Assumption. Let $(G, g, q) \leftarrow \mathsf{GGen}(1^\lambda)$ be a group generator for a multiplicatively written group G of prime-order q with $\log_2 q = \lambda$, generated by g, i.e., $\langle g \rangle = G$. The discrete-logarithm problem (DLP) associated to GGen is to find x when given G, g, q, and g^x with $x \leftarrow \mathbb{Z}_q$. The DL assumption now states that the DLP is hard, i.e., that for every ppt adversary $\mathcal{A}$, there exists a negligible function ν such that:

$$\Pr[(G, g, q) \leftarrow \mathsf{GGen}(1^\lambda), x \leftarrow \mathbb{Z}_q, x' \leftarrow \mathcal{A}(G, g, q, g^x) : x = x'] \leq \nu(\lambda).$$

We sometimes sample from $\mathbb{Z}_q^*$ instead of $\mathbb{Z}_q$. This changes the view of an adversary only negligibly, and is thus not made explicit.

2.2 Building Blocks

Collision-Resistant Hash Function Families. A family $\{\mathcal{H}_\mathcal{R}^k\}_{k \in \mathcal{K}}$ of hash-functions $\mathcal{H}_\mathcal{R}^k : \{0,1\}^* \to \mathcal{R}$ indexed by key $k \in \mathcal{K}$ is collision-resistant if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that:

$$\Pr[k \leftarrow \mathcal{K}, (v, v') \leftarrow \mathcal{A}(k) : \mathcal{H}_\mathcal{R}^k(v) = \mathcal{H}_\mathcal{R}^k(v') \ \wedge \ v \neq v'] \leq \nu(\lambda).$$

Public-Key Encryption Schemes. Public-key encryption allows to encrypt a message m using a given public key pk so that the resulting ciphertext can be decrypted using the corresponding secret key sk. More formally:

Definition 1 (Public-Key Encryption Schemes). *A public-key encryption scheme Π is a triple* $(\mathsf{KGen}_\mathsf{enc}, \mathsf{Enc}, \mathsf{Dec})$ *of ppt algorithms such that:*

$\mathsf{KGen}_\mathsf{enc}$. *The algorithm* $\mathsf{KGen}_\mathsf{enc}$ *on input security parameter λ outputs the private and public keys of the scheme:* $(\mathsf{sk}_\mathsf{enc}, \mathsf{pk}_\mathsf{enc}) \leftarrow \mathsf{KGen}_\mathsf{enc}(1^\lambda)$.

Enc. *The algorithm* Enc *gets as input the public key* pk_enc*, and the message $m \in \mathcal{M}$ and outputs a ciphertext c: $c \leftarrow$* $\mathsf{Enc}(\mathsf{pk}_\mathsf{enc}, m)$.

Dec. *The algorithm* Dec *on input a private key* sk_enc *and a ciphertext c outputs a message $m \in \mathcal{M} \cup \{\bot\}$: $m \leftarrow$* $\mathsf{Dec}(\mathsf{sk}_\mathsf{enc}, c)$.

Definition 2 (Secure Public-Key Encryption Schemes). *We call a public-key encryption scheme Π IND-T secure, if it is correct, and IND-T-secure with $T \in \{CPA, CCA2\}$.*

The formal security definitions are given in App. A.

Non-Interactive Proof Systems. Let L be an **NP**-language with associated witness relation R, i.e., $L = \{x \mid \exists w : R(x, w) = 1\}$. Throughout this paper, we use the Camenisch-Stadler notation [CS97] to express the statements proven in non-interactive, simulation-sound extractable, zero-knowledge (as defined below). In particular, we write $\pi \leftarrow \text{NIZKPoK}\{(w) : R(x, w) = 1\}$ to denote the computation of a non-interactive, simulation-sound extractable, zero-knowledge proof, where all values not in the parentheses are assumed to be public. For example, let L be defined by the following **NP**-relation for a group $(G, g, q) \leftarrow \mathsf{GGen}(1^\lambda)$:

$$((g, h, y, z), (a, b)) \in R \iff y = g^a \ \wedge \ z = g^b h^a.$$

Then, we write $\pi \leftarrow \text{NIZKPoK}\{(a, b) : y = g^a \wedge z = g^b h^a\}$ to denote the corresponding proof of knowledge of witness $(a, b) \in \mathbb{Z}_q^2$ with respect to the statement $(g, h, y, z) \in G^4$. Additionally, we use $\{\texttt{false}, \texttt{true}\} \leftarrow \mathsf{Verify}(x, \pi)$ to denote the corresponding verification algorithm and $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$ to denote the crs generation algorithm. We do not make the crs explicit and, for proof systems where a crs is required, we assume it to be an implicit input to all algorithms.

Definition 3. *We call a* NIZKPoK *secure, if it is complete, simulation-sound extractable, and zero-knowledge.*

The corresponding definitions can be found in App. A.

Chameleon-Hashes. Let us formally define a "standard" chameleon-hash. The framework is based upon the work done by Ateniese et al. and Brzuska et al. [AMVA16, BFF+09], but adapted to fit our notation. Additionally, we provide some extended security definitions.

Definition 4. *A chameleon-hash* CH *consists of five algorithms* ($\mathsf{CParGen}$, CKGen, CHash, $\mathsf{CHashCheck}$, Adapt)*, such that:*

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

CKGen. *The algorithm* CKGen *given the public parameters* $\mathsf{pp_{ch}}$ *outputs the private and public keys of the scheme:* $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{CKGen}(\mathsf{pp_{ch}})$.

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

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

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

Correctness. For a CH we require the correctness property to hold. In particular, we require that for all $\lambda \in \mathbb{N}$, for all $\mathsf{pp_{ch}} \leftarrow \mathsf{CParGen}(1^\lambda)$, for all $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{CKGen}(\mathsf{pp_{ch}})$, for all $m \in \mathcal{M}$, for all $(h, r) \leftarrow \mathsf{CHash}(\mathsf{pk_{ch}}, m)$, for all $m' \in \mathcal{M}$, we have for all for all $r' \leftarrow \mathsf{Adapt}(\mathsf{sk_{ch}}, m, m', r, h)$, that $\texttt{true} = \mathsf{CHashCheck}(\mathsf{pk_{ch}}, m, r, h) = \mathsf{CHashCheck}(\mathsf{pk_{ch}}, m', r', h)$. This definition captures perfect correctness. The randomness is drawn by CHash, and not outside. This was done to capture "private-coin" constructions [AMVA16].

[6] The randomness r is also sometimes called "check value" [AMVA16].

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

Fig. 1. Indistinguishability

Indistinguishability. Indistinguishability requires that the randomnesses r does not reveal if it was obtained through CHash or Adapt. The messages are chosen by the adversary. We relax the perfect indistinguishability definition of Brzuska et al. [BFF+09] to a computational version, which is enough for most use-cases, including ours.

Note that we need to return $\perp$ in the $\mathsf{HashOrAdapt}$ oracle, as the adversary may try to enter a message $m \notin \mathcal{M}$, even if $\mathcal{M} = \{0,1\}^*$, which makes the algorithm output $\perp$. If we would not do this, the adversary could trivially decide indistinguishability. For similar reasons these checks are also included in other definitions.

Definition 5 (Indistinguishability). *A chameleon-hash* CH *is* *indistinguishable, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\left|\Pr[\mathsf{Indistinguishability}_{\mathcal{A}}^{\mathsf{CH}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$ *. 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 an even stronger definition than key-exposure freeness [AdM04b]: key-exposure freeness only requires that one cannot find a collision for some new "tag", i.e., for some auxiliary value for which the adversary has never seen a collision.

Definition 6 (Collision-Resistance). *A chameleon-hash* CH *is* *collision-resistant, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{CollRes}_{\mathcal{A}}^{\mathsf{CH}}(1^\lambda) = 1] \leq \nu(\lambda)$*. The corresponding experiment is depicted in Fig. 2.*

```
Experiment $\mathsf{CollRes}^{\mathsf{CH}}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp_{ch}} \leftarrow \mathsf{CParGen}(1^\lambda)$
  $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{CKGen}(\mathsf{pp_{ch}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{Adapt}'(\mathsf{sk_{ch}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk_{ch}})$
    where oracle $\mathsf{Adapt}'$ on input $\mathsf{sk_{ch}}, m, m', r, h$:
      Return $\perp$, if $\mathsf{CHashCheck}(\mathsf{pk_{ch}}, m, r, h) \neq \mathtt{true}$
      $r' \leftarrow \mathsf{Adapt}(\mathsf{sk_{ch}}, m, m', r, h)$
      If $r' = \perp$, return $\perp$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m, m'\}$
      return $r'$
  return 1, if $\mathsf{CHashCheck}(\mathsf{pk_{ch}}, m^*, r^*, h^*) = \mathsf{CHashCheck}(\mathsf{pk_{ch}}, m'^*, r'^*, h^*) = \mathtt{true} \wedge$
    $m'^* \notin \mathcal{Q} \ \wedge \ m^* \neq m'^*$
  return 0
```

Fig. 2. Collision Resistance

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

Fig. 3. Uniqueness

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

Definition 7 (Uniqueness). *A chameleon-hash* CH *is unique, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{Uniqueness}^{\mathsf{CH}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 3.*

Definition 8 (Secure Chameleon-Hashes). *We call a chameleon-hash* CH *secure, if it is correct, indistinguishable, and collision-resistant.*

We do not consider uniqueness as a fundamental security property, as it depends on the concrete use-case whether this notion is required.

In Appendix D.1, we show how to construct such a unique chameleon-hash, based on the ideas by Brzuska et al. [BFF+09].

3 Chameleon-Hashes with Ephemeral Trapdoors

As already mentioned, a chameleon-hash with ephemeral trapdoor (CHET) allows to prevent the holder of the trapdoor $\mathsf{sk_{ch}}$ from finding collisions, as long as no additional ephemeral trapdoor etd is known. This additional ephemeral trapdoor is chosen freshly for each new hash, and providing, or withholding, this trapdoor thus allows to decide upon each hash computation if finding a collision is possible for the holder of the long-term trapdoor. Hence, we need to introduce a new framework given next, which is also accompanied by suitable security definitions.

Definition 9 (Chameleon-Hashes with Ephemeral Trapdoors). *A chameleon-hash with ephemeral trapdoors* CHET *is a tuple of five algorithms* $(\mathsf{CParGen}, \mathsf{CKGen}, \mathsf{CHash}, \mathsf{CHashCheck}, \mathsf{Adapt})$*, such that:*

$\mathsf{CParGen}$. *The algorithm* $\mathsf{CParGen}$ *on input security parameter* λ *outputs the public parameters:* $\mathsf{pp_{ch}} \leftarrow \mathsf{CParGen}(1^\lambda)$*. For simplicity, we assume that* $\mathsf{pp_{ch}}$ *is an implicit input to all other algorithms.*

CKGen. *The algorithm* CKGen *given the public parameters* $\mathsf{pp_{ch}}$ *outputs the long-term private and public keys of the scheme:* $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{CKGen}(\mathsf{pp_{ch}})$*.*

CHash. *The algorithm* CHash *gets as input the public key* $\mathsf{pk_{ch}}$*, and a message* m *to hash. It outputs a hash* h*, randomness* r*, and the trapdoor information:* $(h, r, \mathsf{etd}) \leftarrow \mathsf{CHash}(\mathsf{pk_{ch}}, m)$*.*

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

Adapt. *The algorithm* Adapt *gets as input* $\mathsf{sk_{ch}}$*, the old message* m*, the old randomness* r*, the new message* m'*, the hash* h*, and the trapdoor information* etd *and outputs new randomness* r'*:* $r' \leftarrow \mathsf{Adapt}(\mathsf{sk_{ch}}, m, m', r, h, \mathsf{etd})$*.*

Correctness. For each CHET we require the correctness properties to hold. In particular, we require that for all security parameters $\lambda \in \mathbb{N}$, for all $\mathsf{pp_{ch}} \leftarrow \mathsf{CParGen}(1^\lambda)$, for all $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{CKGen}(\mathsf{pp_{ch}})$, for all $m \in \mathcal{M}$, for all $(h, r, \mathsf{etd}) \leftarrow \mathsf{CHash}(\mathsf{pk_{ch}}, m)$, we have $\mathsf{CHashCheck}(\mathsf{pk_{ch}}, m, r, h) = \mathtt{true}$, and additionally for all $m' \in \mathcal{M}$, for all $r' \leftarrow \mathsf{Adapt}(\mathsf{sk_{ch}}, m, m', r, h, \mathsf{etd})$, we have $\mathsf{CHashCheck}(\mathsf{pk_{ch}}, m', r', h) = \mathtt{true}$. 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 CHash or Adapt. In other words, an outsider cannot decide whether a message is the original one or not.

Definition 10 (Indistinguishability). *A chameleon-hash with ephemeral trapdoor* CHET *is* indistinguishable*, if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\left|\Pr[\mathsf{Indistinguishability}^{\mathsf{CHET}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$*. The corresponding experiment is depicted in Fig. 4.*

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

```
Experiment $\mathsf{Indistinguishability}^{\mathsf{CHET}}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp_{ch}} \leftarrow \mathsf{CParGen}(1^\lambda)$
  $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{CKGen}(\mathsf{pp_{ch}})$
  $b \leftarrow \{0, 1\}$
  $a \leftarrow \mathcal{A}^{\mathsf{HashOrAdapt}(\mathsf{sk_{ch}},\cdot,\cdot,b),\mathsf{Adapt}(\mathsf{sk_{ch}},\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk_{ch}})$
    where oracle $\mathsf{HashOrAdapt}$ on input $\mathsf{sk_{ch}}, m, m', b$:
      let $(h, r, \mathsf{etd}) \leftarrow \mathsf{CHash}(\mathsf{pk_{ch}}, m')$
      let $(h', r', \mathsf{etd}') \leftarrow \mathsf{CHash}(\mathsf{pk_{ch}}, m)$
      let $r'' \leftarrow \mathsf{Adapt}(\mathsf{sk_{ch}}, m, m', r', h', \mathsf{etd}')$
      if $r'' = \perp \;\vee\; r' = \perp$, return $\perp$
      if $b = 0$:
        return $(h, r, \mathsf{etd})$
      if $b = 1$:
        return $(h', r'', \mathsf{etd}')$
  return 1, if $a = b$
  return 0
```

Fig. 4. Indistinguishability

```
Experiment $\mathsf{PublicCollRes}^{\mathsf{CHET}}_{\mathcal{A}}(\lambda)$
  $\mathsf{pp_{ch}} \leftarrow \mathsf{CParGen}(1^\lambda)$
  $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{CKGen}(\mathsf{pp_{ch}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{Adapt}'(\mathsf{sk_{ch}},\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk_{ch}})$
    where oracle $\mathsf{Adapt}'$ on input $\mathsf{sk_{ch}}, m, m', r, \mathsf{etd}, h$:
      return $\perp$, if $\mathsf{CHashCheck}(\mathsf{pk_{ch}}, m, r, h) = \mathtt{false}$
      $r' \leftarrow \mathsf{Adapt}(\mathsf{sk_{ch}}, m, m', r, h, \mathsf{etd})$
      If $r' = \perp$, return $\perp$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m, m'\}$
      return $r'$
  return 1, if $\mathsf{CHashCheck}(\mathsf{pk_{ch}}, m^*, r^*, h^*) = \mathtt{true} \;\wedge$
    $\mathsf{CHashCheck}(\mathsf{pk_{ch}}, m'^*, r'^*, h^*) = \mathtt{true} \;\wedge$
    $m'^* \notin \mathcal{Q} \;\wedge\; m^* \neq m'^*$
  return 0
```

Fig. 5. Public Collision-Resistance

Definition 11 (Public Collision-Resistance). *A chameleon-hash with ephemeral trapdoor* CHET *is* publicly collision-resistant, *if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{PublicCollRes}^{\mathsf{CHET}}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$*. The corresponding experiment is depicted in Fig. 5.*

Private Collision-Resistance. Private collision resistance requires that even the holder of the secret key $\mathsf{sk_{ch}}$ cannot find collisions as long as etd is unknown. This is formalized by a honest hashing oracle which does not return etd. Hence, $\mathcal{A}$'s goal is to return an actual collision on a non-adversarially generated hash h, for which it does not know etd.

Definition 12 (Private Collision-Resistance). *A chameleon-hash with ephemeral trapdoor* CHET *is* privately collision-resistant, *if for any efficient adversary*

```
Experiment $\mathsf{PrivateCollRes}_{\mathcal{A}}^{\mathsf{CHET}}(\lambda)$
  $\mathsf{pp_{ch}} \leftarrow \mathsf{CParGen}(1^\lambda)$
  $\mathcal{Q} \leftarrow \emptyset$
  $(\mathsf{pk}^*, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{pp_{ch}})$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow \mathcal{A}^{\mathsf{CHash}'(\mathsf{pk}^*, \cdot)}(\mathsf{state})$
    where oracle $\mathsf{CHash}'$ on input $\mathsf{pk}^*, m$:
      $(h, r, \mathsf{etd}) \leftarrow \mathsf{CHash}(\mathsf{pk}^*, m)$
      If $h = \bot$, return $\bot$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(h, m)\}$
      return $(h, r)$
  return 1, if $\mathsf{CHashCheck}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathtt{true} \wedge$
    $\mathsf{CHashCheck}(\mathsf{pk}^*, m'^*, r'^*, h^*) = \mathtt{true} \wedge$
    $(h^*, m^*) \notin \mathcal{Q} \ \wedge \ (h^*, \cdot) \in \mathcal{Q}$
  return 0
```

Fig. 6. Private Collision-Resistance

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

Fig. 7. Uniqueness

$\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{PrivateCollRes}_{\mathcal{A}}^{\mathsf{CHET}}(1^\lambda) = 1] \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 6.*

Uniqueness. Uniqueness requires that it is hard to come up with two different randomness values for the same message m^* and hash value h^*, where pk^* is adversarially chosen.

Definition 13 (Uniqueness). *A chameleon-hash with ephemeral trapdoor* CHET *is* unique, *if for any efficient adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that* $\Pr[\mathsf{Uniqueness}_{\mathcal{A}}^{\mathsf{CHET}}(1^\lambda) = 1] \leq \nu(\lambda)$. *The corresponding experiment is depicted in Fig. 7.*

Definition 14 (Secure Chameleon-Hashes with Ephemeral Trapdoors). *We call a chameleon-hash with ephemeral trapdoor* CHET *secure, if it is correct, indistinguishable, publicly collision-resistant, and privately collision-resistant.*

Note, we do not require that a secure CHET is unique, as it depends on the use-case whether this strong security notion is required.

4 Constructions

Regarding constructions of CHET schemes, we first ask the natural question whether CHETs can be built from existing primitives in a black-box way. Interestingly, we can show how to elegantly "bootstrap" a CHET scheme in a black-box fashion from *any* existing secure (and unique) chameleon-hash. Since, however, a secure chameleon-hash does not exist to date, we show how to construct it in App. D.1, based on the ideas by Brzuska et al. [BFF+09]. If one does not require uniqueness, one can, e.g., resort to the recent scheme given by Ateniese et al. [AMVA16].

We then proceed in presenting three direct constructions, two based on the DL assumption, and one based on an RSA-like assumption. While the DL-based constructions are not unique, the construction from RSA-like assumptions even achieves uniqueness. We however note that this strong security notion is not required in all use-cases. For example, in our application scenario (cf. Section 5), the CHETs do not need to be unique.

4.1 Black-Box Construction: Bootstrapping

We now present a black-box construction from any existing chameleon-hash. Namely, we show how one can achieve our desired goals by combining two instances of a secure chameleon-hash CH.

Construction 1 (Bootstrapped Construction) *We omit obvious checks for brevity. Let* CHET *be defined as:*

$\mathsf{CParGen}$. *The algorithm* $\mathsf{CParGen}$ *does the following:*
  1. *Return* $\mathsf{pp_{ch}} \leftarrow \mathsf{CH.CParGen}(1^\lambda)$.

CKGen. *The algorithm* CKGen *generates the key pair in the following way:*
  1. *Return* $(\mathsf{sk}_{\mathsf{ch}}^1, \mathsf{pk}_{\mathsf{ch}}^1) \leftarrow \mathsf{CH.CKGen}(\mathsf{pp_{ch}})$.

CHash. *To hash a message* m*, w.r.t. public key* $\mathsf{pk}_{\mathsf{ch}}^1$ *do:*
  1. *Let* $(\mathsf{sk}_{\mathsf{ch}}^2, \mathsf{pk}_{\mathsf{ch}}^2) \leftarrow \mathsf{CH.CKGen}(\mathsf{pp_{ch}})$.
  2. *Let* $(h_1, r_1) \leftarrow \mathsf{CH.CHash}(\mathsf{pk}_{\mathsf{ch}}^1, m)$.
  3. *Let* $(h_2, r_2) \leftarrow \mathsf{CH.CHash}(\mathsf{pk}_{\mathsf{ch}}^2, m)$.
  4. *Return* $((h_1, h_2, \mathsf{pk}_{\mathsf{ch}}^2), (r_1, r_2), \mathsf{sk}_{\mathsf{ch}}^2)$.

$\mathsf{CHashCheck}$. *To check whether a given hash* $h = (h_1, h_2, \mathsf{pk}_{\mathsf{ch}}^2)$ *is valid on input* $\mathsf{pk_{ch}} = \mathsf{pk}_{\mathsf{ch}}^1$*,* m*,* $r = (r_1, r_2)$*, do:*
  1. *Let* $b_1 \leftarrow \mathsf{CH.CHashCheck}(\mathsf{pk}_{\mathsf{ch}}^1, m, r_1, h_1)$.
  2. *Let* $b_2 \leftarrow \mathsf{CH.CHashCheck}(\mathsf{pk}_{\mathsf{ch}}^2, m, r_2, h_2)$.
  3. *If* $b_1 = \mathtt{false} \ \vee \ b_2 = \mathtt{false}$*, return* $\mathtt{false}$.
  4. *Return* $\mathtt{true}$.

Adapt. *To find a collision w.r.t.* m*,* m'*, randomness* $r = (r_1, r_2)$*, hash* $h = (h_1, h_2, \mathsf{pk}_{\mathsf{ch}}^2)$*,* $\mathsf{etd} = \mathsf{sk}_{\mathsf{ch}}^2$*, and* $\mathsf{sk_{ch}} = \mathsf{sk}_{\mathsf{ch}}^1$ *do:*
  1. *If* $\mathtt{false} = \mathsf{CHashCheck}(\mathsf{pk_{ch}}, m, r, h)$*, return* $\perp$.
  2. *Compute* $r_1' \leftarrow \mathsf{CH.Adapt}(\mathsf{sk}_{\mathsf{ch}}^1, m, m', r_1, h_1)$.
  3. *Compute* $r_2' \leftarrow \mathsf{CH.Adapt}(\mathsf{sk}_{\mathsf{ch}}^2, m, m', r_2, h_2)$.
  4. *Return* (r_1', r_2').

The proof of the following theorem can be found in App. F.1.

Theorem 1. *If* CH *is secure and unique, then the chameleon-hash with ephemeral trapdoors* CHET *in Construction 1 is secure, and unique.*

This construction is easy to understand and only uses standard primitives. The question is now, if we can also directly construct CHET, which we answer to the affirmative subsequently.

4.2 A First Direct Construction

We now present a direct construction in groups where the DLP is hard using some ideas related to Pedersen commitments [Ped91]. In a nutshell, the long-term secret is the discrete logarithm x between two elements g and h (i.e., $g^x = h$) of the long-term public key, while the ephemeral trapdoor is the randomness of the "commitment". To prohibit that a seen collision allows to extract the long-term secret key x, both trapdoors are hidden in a NIZKPoK. To make the "commitment" equivocable, it is then again randomized. To avoid that the holder of $\mathsf{sk_{ch}}$ needs to store state, the randomness is encrypted to a public key of a IND-CCA2 secure encryption scheme contained in $\mathsf{pk_{ch}}$. Security then directly follows from the DL assumption, IND-CCA2, the collision-resistance of the used hash function, and the extractability property of the NIZKPoK system. For brevity we assume that the **NP**-languages involved in the NIZKPoKs are implicitly defined by the scheme. Note, this construction is not unique.

Construction 2 (CHET in Known-Order Groups) *Let* $\{\mathcal{H}^k_{\mathbb{Z}^*_q}\}_{k\in\mathcal{K}}$ *denote a family of collision-resistant hash functions* $\mathcal{H}^k_{\mathbb{Z}^*_q} : \{0,1\}^* \to \mathbb{Z}^*_q$ *indexed by a key* $k \in \mathcal{K}$ *and let* CHET *be as follows:*

CParGen. *The algorithm* CParGen *generates the public parameters in the following way:*

1. *Let* $(G, g, p) \leftarrow \mathsf{GGen}(1^\lambda)$.
2. *Let* $k \leftarrow \mathcal{K}$ *for the hash function.*
3. *Let* $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$.[7]
4. *Return* $((G, g, q), k, \mathsf{crs})$.

CKGen. *The algorithm* CKGen *generates the key pair in the following way:*

1. *Draw random* $x \leftarrow \mathbb{Z}^*_q$. *Set* $h \leftarrow g^x$.
2. *Generate* $\pi_{\mathsf{pk}} \leftarrow \mathrm{NIZKPoK}\{(x) : h = g^x\}$.
3. *Let* $(\mathsf{sk_{enc}}, \mathsf{pk_{enc}}) \leftarrow \mathit{\Pi}.\mathsf{KGen_{enc}}(1^\lambda)$.
4. *Return* $((x, \mathsf{sk_{enc}}), (h, \pi_{\mathsf{pk}}, \mathsf{pk_{enc}}))$.

CHash. *To hash* m *w.r.t.* $\mathsf{pk_{ch}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk_{enc}})$ *do:*

1. *Return* $\perp$, *if* $h \notin G^*$.
2. *If* π_{pk} *is not valid, return* $\perp$.
3. *Draw random* $r \leftarrow \mathbb{Z}^*_q$.
4. *Draw random* $\mathsf{etd} \leftarrow \mathbb{Z}^*_q$.
5. *Let* $h' \leftarrow g^{\mathsf{etd}}$.
6. *Generate* $\pi_t \leftarrow \mathrm{NIZKPoK}\{(\mathsf{etd}) : h' = g^{\mathsf{etd}})\}$.
7. *Encrypt* r, *i.e., let* $C \leftarrow \mathit{\Pi}.\mathsf{Enc}(\mathsf{pk_{enc}}, r)$.
8. *Let* $a \leftarrow \mathcal{H}^k_{\mathbb{Z}^*_q}(m)$.
9. *Let* $p \leftarrow h^r$.
10. *Generate* $\pi_p \leftarrow \mathrm{NIZKPoK}\{(r) : p = h^r\}$.
11. *Let* $b \leftarrow ph'^a$.
12. *Return* $((b, h', \pi_t), (p, C, \pi_p), \mathsf{etd})$.

[7] Actually we need one crs per language, but we do not make this explicit here.

$\mathsf{CHashCheck}$. *To check whether a given hash* (b, h', π_t) *is valid on input* $\mathsf{pk_{ch}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk_{enc}}), m, r = (p, C, \pi_p)$, *do:*

1. *Return* `false`*, if* $p \notin G^* \ \vee \ h' \notin G^*$.
2. *If either* π_p, π_t, *or* π_{pk} *are not valid, return* $\perp$.
3. *Let* $a \leftarrow \mathcal{H}^k_{\mathbb{Z}^*_q}(m)$.
4. *Return* `true`*, if* $b = ph'^a$.
5. *Return* `false`.

Adapt. *To find a collision w.r.t.* m, m', (b, h', π_t), *randomness* (p, C, π_p), *and trapdoor information* etd, *and* $\mathsf{sk_{ch}} = (x, \mathsf{sk_{enc}})$ *do:*

1. *If* `false` $= \mathsf{CHashCheck}(\mathsf{pk_{ch}}, m, (p, C, \pi_p), (b, h', \pi_t))$, *return* $\perp$.
2. *Decrypt* C, *i.e.,* $r \leftarrow \Pi.\mathsf{Dec}(\mathsf{sk_{enc}}, C)$. *If* $r = \perp$, *return* $\perp$.
3. *If* $h' \neq g^{\mathsf{etd}}$, *return* $\perp$.
4. *Let* $a \leftarrow \mathcal{H}^k_{\mathbb{Z}^*_q}(m)$.
5. *Let* $a' \leftarrow \mathcal{H}^k_{\mathbb{Z}^*_q}(m')$.
6. *If* $p \neq g^{xr}$, *return* $\perp$.
7. *If* $a = a'$, *return* (p, C, π_p).
8. *Let* $r' \leftarrow \frac{rx + a \cdot \mathsf{etd} - a' \cdot \mathsf{etd}}{x}$.
9. *Let* $p' \leftarrow h^{r'}$.
10. *Encrypt* r', *i.e., let* $C' \leftarrow \Pi.\mathsf{Enc}(\mathsf{pk_{enc}}, r')$.
11. *Generate* $\pi'_p \leftarrow \mathrm{NIZKPoK}\{(r') : p' = h^{r'}\}$.
12. *Return* (p', C', π'_p).

Some of the checks can already be done in advance, e.g., at a PKI, which only generates certificates, if the restrictions on each public key are fulfilled.

The proof of the following Theorem is given in App. F.3.

Theorem 2. *If the DL assumption in* G *holds,* $\mathcal{H}^k_{\mathbb{Z}^*_{|G|}}$ *is collision-resistant,* Π *is IND-CCA2 secure, and* NIZKPoK *is secure, then the chameleon-hash with ephemeral trapdoors* CHET *in Construction 2 is secure.*

Two further constructions, one based on the DL assumption in gap-groups, and one based on RSA-like assumptions (in the random oracle model, which is also unique), are given in App. E.

5 Application: Invisible Sanitizable Signatures

Informally, security of digital signatures requires that a signature σ on a message m becomes invalid as soon as a single bit of m is altered [GMR88]. However, there are many real-life use-cases in which a subsequent change to signed data by a semi-trusted party without invalidating the signature is desired. As a simplified example, consider a patient record which is signed by a medical doctor. The accountant, which charges the insurance company, only requires knowledge of the treatments and the patient's insurance number. This protects the patient's privacy. In this constellation, having the data re-signed by the M.D. whenever subsets of the record need to be forwarded to some party induces too much overhead to be practical in real scenarios or may even be impossible due to availability constraints.

Sanitizable signature schemes (SSS) [ACdMT05] address these shortcomings. They allow the signer to determine which blocks $m[i]$ of a given message $m = (m[1], m[2], \ldots, m[i], \ldots, m[\ell])$ are admissible. Any such admissible block can be changed to a different bitstring $m[i]' \in \{0,1\}^*$, where $i \in \{1, 2, \ldots, \ell\}$, by a semi-trusted party named the sanitizer. This party is identified by a private/public key pair and the sanitization process described before requires the private key. 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 under the original public key. Thus, authenticity of the message is still ensured. In the prior example, for the server storing the data it is possible to already black-out the sensitive parts of a signed document without any additional communication with the M.D. and in particular without access to the signing key of the M.D.

Real-world applications of SSSs include the already mentioned privacy-preserving handling of patient data, secure routing, privacy-preserving document disclosure, credentials, and blank signatures [ACdMT05, BFLS10, BPS12, BPS13, CL13, DHS14, HS13].

Our Contribution. We introduce the notion of *invisible* SSSs. This strong privacy notion requires that a third party not holding any secret keys cannot decide whether a specific block is admissible, i.e., can be sanitized. This has already been discussed by Ateniese et al. [ACdMT05] in the first work on sanitizable signatures, but they neither provide a formal framework nor a provably secure construction. However, we identify some use-cases where such a notion is important, and we close this gap by introducing a new framework for SSSs, along with an extended security model. Moreover, we propose a construction being provably secure in our framework. Our construction paradigm is based on IND-CPA secure encryption schemes, standard, yet unique, chameleon-hashes, and strongly unforgeable signature schemes. These can be considered standard tools nowadays. We pair those with a chameleon-hash with ephemeral trapdoors.

Motivation. At PKC '09, Brzuska et al. formalized the most common security model of SSSs [BFF$^+$09]. For our work, the most important property they are addressing is "weak transparency". It means that although a third party sees which blocks of a message are admissible, it cannot decide whether some block has already been sanitized by a sanitizer. More precisely, their formalization explicitly requires that the third party is always able to decide whether a given block in a message is admissible. However, as this may invade privacy, having a construction which hides this additional information is useful as well. To address this problem the notion of "strong transparency" has been informally proposed in the original work by Ateniese et al. [ACdMT05].

Examples. To make the usefulness of such a stronger privacy property more visible, consider the following two application scenarios.

In the first scenario, we consider that a document is the output of a workflow that requires several—potentially heavy—computations to become ready. We assume that the output of each workflow step could be produced by one party alone, but could also be outsourced. However, if the party decides to outsource the production of certain parts of the document it wants the potential involvement of other parties to stay hidden, e.g., the potential and actual outsourcing might be considered a trade secret. In order to regain some control that all tasks are done only by authorized subordinates, the document—containing template parts—is signed with a sanitizable signature. Such an approach, i.e., to use SSS for workflow control, was proposed in [DHPS15].

The second one is motivated by an ongoing legal debate in Germany.[8] Consider a school class where a pupil suffers from dyslexia[9] and thus can apply for additional help to compensate the illness. One way to compensate this is to consider spelling mistakes less when giving grades. Assume that only the school's principal shall decide to what extent a certain grade shall be improved. Of course, this shall only be possible for pupils who are actually handicapped. For the pupil with dyslexia, e.g., known to the teacher of the class in question, the grade is marked as sanitizable by the principal. The legal debate in Germany is about an outsider, e.g., future employer, who should not be able to decide that grades had the potential to be altered and of course also not see for which pupils the grades have been altered to preserve their privacy. To achieve this, standard sanitizable signature schemes are clearly not enough, as they do not guarantee that an outsider cannot derive which blocks are potentially sanitizable, i.e., which pupil is actually handicapped. We offer a solution to this problem, where an outsider cannot decide which block is admissible, i.e., can be altered.

State-of-the-Art. SSSs have been introduced by Ateniese et al. [ACdMT05]. Brzuska et al. formalized most of the current security properties [BFF$^+$09]. These have been later extended for (strong) unlinkability [BFLS10,BPS13,FKM$^+$16] and non-interactive public accountability [BPS12, BPS13]. Some properties discussed by Brzuska et al. [BFF$^+$09] have then been refined by Gong et al. [GQZ11]. Namely, they also consider the admissible blocks in the security games, while still requiring that these are visible to everyone. Recently, Krenn et al. further refined the security properties to also account for the signatures, not only the message [KSS15].[10] We use the aforementioned results as our starting point for the extended definitions.

[8] See for example the ruling from the German Federal Administrative Court (BVerwG) 29.07.2015, Az.: 6 C 33.14, 6 C 35.14.

[9] A disorder involving difficulty in learning to read or interpret words, letters and other symbols.

[10] We want to stress that Krenn et al. [KSS15] also introduce "strong transparency", which is not related to the definition given by Ateniese et al. [ACdMT05].

Also, several extensions such as limiting the sanitizer to signer-chosen values [CJ10, DS15, KL06, PSP11], trapdoor SSSs (which allow to add new sanitizers after signature generation by the signer) [CLM08, YSL10], multi-sanitizer and -signer environments for SSSs [BFLS09, BPS13, CJL12], and sanitization of signed and encrypted data [FF15] have been considered. SSSs have also been used as a tool to make other primitives accountable [PS15], and to build other primitives [BHPS16, dMPPS14]. Also, SSSs and data-structures being more complex than lists have been considered [PSP11]. Our results carry over to the aforementioned extended settings with only minor additional adjustments. Implementations of SSSs have also been presented [BPS12, BPS13, dMPPS13, PPS+13].

Of course, computing on signed messages is a broad field. We can therefore only give a small overview. Decent and comprehensive overviews of other related primitives, however, have already been published [ABC+12, BBD+10, DDH+15, GGOT15, GOT15].

5.1 Additional Building Blocks

We assume that the reader is familiar with digital signatures, PRGs, and PRFs, and only introduce the notation used in the following. A PRF consists of a key generation algorithm $\mathsf{KGen}_{\mathsf{prf}}$ and an evaluation algorithm $\mathsf{Eval}_{\mathsf{prf}}$; similarly, a PRG consists of an evaluation algorithm $\mathsf{Eval}_{\mathsf{prg}}$. Finally, a digital signature scheme Σ consists of a key generation algorithm $\mathsf{KGen}_{\mathsf{sig}}$, a signing algorithm Sign, and a verification algorithm Verify. For formal definitions and the required security notions, cf. App. A.

5.2 Our Framework for Sanitizable Signature Schemes

Subsequently, we introduce our framework for SSSs. Our definitions are based on existing work [BFF+09, BPS12, BPS13, GQZ11, KSS15]. However, due to our goals, we need to modify the current framework to account for the fact that the admissible blocks are only visible to the sanitizer. We do not consider "non-interactive public accountability" [BPS12, BPS13, HPS12], which allows a third party to decide which party is accountable, as transparency is mutually exclusive to this property, but is very easy to achieve, e.g., by signing the sanitizable signature again [BPS12]. For the sake of completeness, the definitions which are omitted in the following are provided in App. B.

Before we present the formal definition, we settle some notation. The variable ADM contains the set of indices of the modifiable blocks, as well as the number ℓ of blocks in a message m. We write $\mathrm{ADM}(m) = \mathtt{true}$, if ADM is valid w.r.t. m, i.e., ADM contains the correct ℓ and all indices are in m. For example, let $\mathrm{ADM} = (\{1, 2, 4\}, 4)$. 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 $\mathrm{MOD}(\mathrm{ADM}) = \mathtt{true}$, 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 $\widetilde{X}_{n,m}$ with $n \leq m$ for the vector $(X_n, X_{n+1}, X_{n+2}, \ldots, X_{m-1}, X_m)$.

Definition 15 (Sanitizable Signatures). *A sanitizable signature scheme* SSS *consists of eight ppt algorithms* $(\mathsf{SSSParGen}, \mathsf{KGen}_{\mathsf{sig}}, \mathsf{KGen}_{\mathsf{san}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Verify}, \mathsf{Proof}, \mathsf{Judge})$ *such that*

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

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

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

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

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

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

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

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

Correctness of Sanitizable Signature Schemes. We require the usual correctness requirements to hold. In a nutshell, every signed and sanitized message/signature pair should verify, while a honestly generated proof on a honestly generated message/signature pair should point to the correct accountable party. We refer to [BFF+09] for a formal definition, which straightforwardly extends to our framework.

5.3 Security of Sanitizable Signature Schemes

Next, we introduce our security model, where our definitions already incorporate newer insights [BFF$^+$09,BPS13,GQZ11,KSS15]. In particular, we mostly consider the "strong" definitions by Krenn et al. [KSS15] as the new state-of-the-art. Due to our goals, we also see the data-structure corresponding to the admissible blocks, i.e., ADM, as an asset which needs protection, which addresses the work done by Gong et al. [GQZ11]. All formal definitions can be found in App. B.

Unforgeability. No one should be able to generate any new signature not seen before without having access to any private keys.

Immutability. Sanitizers must only be able to perform allowed modifications. In particular, a sanitizer must not be able to modify non-admissible blocks.

Privacy. Similar to semantic security for encryption schemes, privacy captures the inability of an attacker to derive any knowledge about sanitized parts.

Transparency. An attacker cannot tell whether a specific message/signature pair has been sanitized or not.

Accountability. For signer-accountability, a signer should not be able to accuse a sanitizer if the sanitizer is actually not responsible for a given message, and vice versa for sanitizer-accountability.

5.4 Invisibility of SSSs

Next, we introduce the new property of invisibility. Basically, invisibility requires that an outsider cannot decide which blocks of a given message are admissible. With $\mathrm{ADM}_0 \cap \mathrm{ADM}_1$, we denote the intersection of the admissible blocks, ignoring the length of the messages.

In a nutshell, the adversary can query an LoRADM oracle which either makes ADM_0 or ADM_1 admissible in the final signature. Of course, the adversary has to be restricted to $\mathrm{ADM}_0 \cap \mathrm{ADM}_1$ for sanitization requests for signatures originating from those created by LoRADM and their derivatives to avoid trivial attacks. We stress that our invisibility definition is very strong, as it also takes the signatures into account, much like the definitions given by Krenn et al. [KSS15]. Note, the signing oracle can be simulated by LoRADM, setting $\mathrm{ADM}_0 = \mathrm{ADM}_1$. One can easily alter our definition to only account for the messages in question, e.g., if one wants to avoid strongly unforgeable signatures, or even allow re-randomizable signatures. An adjustment is straightforward.

Definition 16 (Invisibility). *An SSS is invisible, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that $\left|\Pr[\mathsf{Invisibility}_{\mathcal{A}}^{\mathsf{SSS}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$, where the corresponding experiment is defined in Fig. 8.*

It is obvious that invisibility is not implied by any other property. In a nutshell, taking any secure SSS, it is sufficient to non-malleably append ADM to each block $m[i]$ to prevent invisibility. Clearly, all other properties of such a construction are still preserved.

We also stress that this definition only ensures security against outsiders, i.e., where no signature under an adversarially chosen signer key is sanitized.

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

Fig. 8. Invisibility

Definition 17 (Secure SSS). *We call an* SSS *secure, if it is correct, private, unforgeable, immutable, sanitizer-accountable, signer-accountable, and invisible.*

We do neither consider non-interactive public accountability nor unlinkability nor transparency as essential security requirements, as it depends on the concrete use-case whether these properties are required.

5.5 Construction

We now introduce our construction and use the construction paradigm of Ateniese et al. [ACdMT05], enriching it with several ideas of prior work [BFF+09, GQZ11, dMPPS13]. The main idea is to hash each block using a chameleon-hash with ephemeral trapdoors, and then sign the hashes. The main trick we introduce to limit the sanitizer is that only those etd_i are given to the sanitizer, for which the respective block $m[i]$ should be sanitizable. To hide whether a given block is sanitizable, each etd_i is encrypted; a sanitizable block contains the real etd_i, while a non-admissible block encrypts a 0, where 0 is assumed to be an invalid etd. For simplicity, we require that the IND-CPA secure encryption scheme Π allows that each possible etd, as well as 0, is in the message space $\mathcal{M}$ of Π, which can be achieved using standard embedding and padding techniques, or using KEM/DEM combinations [AGK08]. To achieve accountability, we generate additional "tags" for a "standard" chameleon-hash (which binds everything together) in a special way, namely we use PRFs and PRGs, which borrows ideas from the construction given by Brzuska et al. [BFF+09].

Construction 3 (Secure and Transparent SSS) *The secure and transparent* SSS *construction is as follows:*

$\mathsf{SSSParGen}$. *To generate the public parameters, do the following steps:*

1. *Let* $\mathsf{pp_{ch}} \leftarrow \mathsf{CHET.CParGen}(1^\lambda)$.
2. *Let* $\mathsf{pp'_{ch}} \leftarrow \mathsf{CH.CParGen}(1^\lambda)$.
3. *Return* $\mathsf{pp_{sss}} = (\mathsf{pp_{ch}}, \mathsf{pp'_{ch}})$.

$\mathsf{KGen_{sig}}$. *To generate the key pair for the signer, do the following steps:*

1. *Let* $(\mathsf{pk_s}, \mathsf{sk_s}) \leftarrow \Sigma.\mathsf{KGen_{sig}}(1^\lambda)$.
2. *Pick a key for a* PRF*, i.e.,* $\kappa \leftarrow \mathsf{PRF.KGen_{prf}}(1^\lambda)$.
3. *Return* $((\kappa, \mathsf{sk_s}), \mathsf{pk_s})$.

$\mathsf{KGen_{san}}$. *To generate the key pair for the sanitizer, do the following steps:*

1. *Let* $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}}) \leftarrow \mathsf{CHET.CKGen}(\mathsf{pp_{ch}})$.
2. *Let* $(\mathsf{sk'_{ch}}, \mathsf{pk'_{ch}}) \leftarrow \mathsf{CH.CKGen}(\mathsf{pp'_{ch}})$.
3. *Let* $(\mathsf{sk_{enc}}, \mathsf{pk_{enc}}) \leftarrow \Pi.\mathsf{KGen_{enc}}(1^\lambda)$.
4. *Return* $((\mathsf{sk_{ch}}, \mathsf{sk'_{ch}}, \mathsf{sk_{enc}}), (\mathsf{pk_{ch}}, \mathsf{pk'_{ch}}, \mathsf{pk_{enc}}))$.

Sign. *To generate a signature* σ*, on input of* $m = (m[1], m[2], \ldots, m[\ell])$, $\mathsf{sk_{sig}} = (\kappa, \mathsf{sk_s})$, $\mathsf{pk_{san}} = (\mathsf{pk_{ch}}, \mathsf{pk'_{ch}}, \mathsf{pk_{enc}})$*, and* ADM *do the following steps:*

1. *If* $\mathrm{ADM}(m) \neq \mathtt{true}$*, return* $\perp$.
2. *Draw* $x_0 \leftarrow \{0,1\}^\lambda$.
3. *Let* $x'_0 \leftarrow \mathsf{PRF.Eval_{prf}}(\kappa, x_0)$.
4. *Let* $\tau \leftarrow \mathsf{PRG.Eval_{prg}}(x'_0)$.
5. *For each* $i \in \{1, 2, \ldots, \ell\}$ *do:*
   - (a) *Set* $(h_i, r_i, \mathsf{etd}_i) \leftarrow \mathsf{CHET.CHash}(\mathsf{pk_{ch}}, (i, m[i], \mathsf{pk_{sig}}))$.
   - (b) *If block* i *is not admissible, let* $\mathsf{etd}_i \leftarrow 0$.
   - (c) *Compute* $c_i \leftarrow \Pi.\mathsf{Enc}(\mathsf{pk_{enc}}, \mathsf{etd}_i)$.
6. *Set* $(h_0, r_0) \leftarrow \mathsf{CH.CHash}(\mathsf{pk'_{ch}}, (0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk_{sig}}))$.
7. *Set* $\sigma' \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk_s}, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk_{san}}, \mathsf{pk_{sig}}, \ell))$.
8. *Return* $\sigma = (\sigma', x_0, \widetilde{r}_{0,\ell}, \tau, \widetilde{c}_{1,\ell}, \widetilde{h}_{0,\ell})$.

Verify. *To verify a signature* $\sigma = (\sigma', x_0, \widetilde{r}_{0,\ell}, \tau, \widetilde{c}_{1,\ell}, \widetilde{h}_{0,\ell})$*, on input of* $m = (m[1], m[2], \ldots, m[\ell])$*, w.r.t. to* $\mathsf{pk_{sig}} = \mathsf{pk_s}$ *and* $\mathsf{pk_{san}} = (\mathsf{pk_{ch}}, \mathsf{pk'_{ch}}, \mathsf{pk_{enc}})$*, do:*

1. *For each* $i \in \{1, 2, \ldots, \ell\}$ *do:*
   - (a) *Set* $b_i \leftarrow \mathsf{CHET.CHashCheck}(\mathsf{pk_{ch}}, (i, m[i], \mathsf{pk_{sig}}), r_i, h_i)$. *If any* $b_i = \mathtt{false}$*, return* $\mathtt{false}$.
2. *Let* $b_0 \leftarrow \mathsf{CH.CHashCheck}(\mathsf{pk'_{ch}}, (0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk_{sig}}), r_0, h_0)$.
3. *If* $b_0 = \mathtt{false}$*, return* $\mathtt{false}$.
4. *Return* $d \leftarrow \Sigma.\mathsf{Verify}(\mathsf{pk_s}, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk_{san}}, \mathsf{pk_{sig}}, \ell), \sigma')$.

Sanit. *To sanitize a signature* $\sigma = (\sigma', x_0, \widetilde{r}_{0,\ell}, \tau, \widetilde{c}_{1,\ell}, \widetilde{h}_{0,\ell})$*, on input of* $m = (m[1], m[2], \ldots, m[\ell])$*, w.r.t. to* $\mathsf{pk_{sig}} = \mathsf{pk_s}$, $\mathsf{sk_{san}} = (\mathsf{sk_{ch}}, \mathsf{sk'_{ch}}, \mathsf{sk_{enc}})$*, and* MOD *do:*

1. *Verify the signature, i.e., run* $d \leftarrow \mathsf{SSS.Verify}(m, \sigma, \mathsf{pk_{sig}}, \mathsf{pk_{san}})$. *If* $d = \mathtt{false}$*, return* $\perp$.
2. *Decrypt each* c_i *for* $i \in \{1, 2, \ldots, \ell\}$*, i.e., let* $\mathsf{etd}_i \leftarrow \Pi.\mathsf{Dec}(\mathsf{sk_{enc}}, c_i)$. *If any decryption fails, return* $\perp$.
3. *For each index* $i \in \mathrm{MOD}$ *check that* $\mathsf{etd}_i \neq 0$. *If not, return* $\perp$.
4. *For each block* $m[i]' \in \mathrm{MOD}$ *do:*

    (a) *Let* $r'_i \leftarrow \mathsf{CHET.Adapt}(\mathsf{sk}_{\mathsf{ch}}, (i, m[i], \mathsf{pk}_{\mathsf{sig}}), (i, m[i]', \mathsf{pk}_{\mathsf{sig}}), r_i, h_i, \mathsf{etd}_i)$.
    (b) *If* $r'_i = \perp$, *return* $\perp$.
5. *For each block* $m[i]' \notin \mathrm{MOD}$ *do:*
    (a) *Let* $r'_i \leftarrow r_i$.
6. *Let* $m' \leftarrow \mathrm{MOD}(m)$.
7. *Draw* $\tau' \leftarrow \{0,1\}^{2\lambda}$.
8. *Let* $r'_0 \leftarrow \mathsf{CH.Adapt}(\mathsf{sk}'_{\mathsf{ch}}, (0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}), (0, m', \tau', \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r'}_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}), r_0, h_0)$.
9. *Return* $(m', (\sigma', x_0, \widetilde{r'}_{0,\ell}, \tau', \widetilde{c}_{1,\ell}, \widetilde{h}_{0,\ell}))$.

Proof. *To create a proof* π, *on input of* $m = (m[1], m[2], \ldots, m[\ell])$, *a signature* σ, *w.r.t. to* $\mathsf{pk}_{\mathsf{san}}$ *and* $\mathsf{sk}_{\mathsf{sig}}$, *and* $\{(m_i, \sigma_i) \mid i \in \mathbb{N}\}$ *do:*

1. *Return* $\perp$, *if* `false` $= \mathsf{SSS.Verify}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$.
2. *Verify each signature in the list, i.e., run* $d_i \leftarrow \mathsf{SSS.Verify}(m_i, \sigma_i, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$. *If for any* $d_i =$ `false`, *return* $\perp$.
3. *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* `true` $= \mathsf{CH.CHashCheck}(\mathsf{pk}'_{\mathsf{ch}}, (0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}), r_0, h_0)$, *and* `true` $= \mathsf{CH.CHashCheck}(\mathsf{pk}'_{\mathsf{ch}}, (0, m', \tau', \ell, \widetilde{h'}_{1,\ell}, \widetilde{c'}_{1,\ell}, \widetilde{r'}_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}), r'_0, h'_0)$ *for some different tag* τ' *or message* m'. *Let this signature/message pair be* $(\sigma', m') \in \{(m_i, \sigma_i) \mid i \in \mathbb{N}\}$.
4. *Return* $\pi = ((\sigma', m'), \mathsf{PRF.Eval}_{\mathsf{prf}}(\kappa, x_0))$, *where* x_0 *is contained in* (σ, m).

Judge. *To find the accountable party on input of* $m = (m[1], m[2], \ldots, m[\ell])$, *a valid signature* σ, *w.r.t. to* $\mathsf{pk}_{\mathsf{san}}$, $\mathsf{pk}_{\mathsf{sig}}$, *and a proof* π *do:*

1. *Check if* π *is of the form* $((\sigma', m'), v)$ *with* $v \in \{0,1\}^{\lambda}$. *If not, return* Sig.
2. *Also return* $\perp$, *if* `false` $= \mathsf{SSS.Verify}(m', \sigma', \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$, *or* `false` $= \mathsf{SSS.Verify}(m, \sigma, \mathsf{pk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{san}})$.
3. *Let* $\tau'' \leftarrow \mathsf{PRG.Eval}_{\mathsf{prg}}(v)$.
4. *If* $\tau' \neq \tau''$, *return* Sig, *where* τ' *is taken from* (σ', m')
5. *If we have a non-trivial collision* $h_0 = h'_0$, `true` $= \mathsf{CH.CHashCheck}(\mathsf{pk}'_{\mathsf{ch}}, (0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}_{\mathsf{sig}}), r_0, h_0) = \mathsf{CH.CHashCheck}(\mathsf{pk}'_{\mathsf{ch}}, (0, m', \tau', \ell', \widetilde{h'}_{1,\ell'}, \widetilde{c'}_{1,\ell'}, \mathsf{pk}_{\mathsf{sig}}), r'_0, h'_0)$, $\widetilde{c}_{1,\ell} = \widetilde{c'}_{1,\ell'}$, $x_0 = x'_0$, $\ell = \ell'$, *and* $\widetilde{h}_{0,\ell} = \widetilde{h'}_{0,\ell'}$, *return* San.
6. *Return* Sig.

Theorem 3. *If Π is IND-CPA secure, Σ,* PRF, PRG, CHET *are secure,* CH *is secure and unique, Construction 3 is a secure and transparent* SSS.

Note, CHET is not required to be unique. We prove each property on its own.

Proof. Correctness follows by inspection.

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

Game 0: The original unforgeability game.

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

Transition - Game 0 → Game 1: Clearly, if $(\sigma'^*, (x_0^*, \widetilde{h}^*_{0,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \mathsf{pk}^*_{\mathsf{san}}, \mathsf{pk}^*_{\mathsf{sig}}, \ell))$ was never obtained by the challenger, this tuple breaks the strong unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a strong unforgeability challenger and embed it as $\mathsf{pk}_{\mathsf{sig}}$. For every required "inner" signature σ', 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 $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{unf\text{-}cma}}(\lambda)$.

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

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

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

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

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

Immutability. We prove immutability using a sequence of games.

Game 0: The immutability game.

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

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. Clearly, if $(\sigma'^*, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}_{\mathsf{san}}, \mathsf{pk}_{\mathsf{sig}}, \ell))$ was never obtained by the challenger, this tuple breaks the strong unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a strong unforgeability challenger and embed it as $\mathsf{pk}_{\mathsf{sig}}$. For every required "inner" signature σ', 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 $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{unf\text{-}cma}}(\lambda)$.

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

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

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

Privacy. We prove privacy; we use a sequence of games.

Game 0: The original privacy game.

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

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

Game 2: As Game 1, but instead of hashing the blocks $(i, m_b[i], \mathsf{pk}_{\mathsf{sig}})$ for the inner chameleon-hashes using CHash, and then Adapt to $(i, m[i], \mathsf{pk}_{\mathsf{sig}})$, we directly apply CHash to $(i, m[i], \mathsf{pk}_{\mathsf{sig}})$.

Transition - Game 1 → Game 2: Assume that the adversary can distinguish this hop. We can then construct an $\mathcal{B}$ which wins the indistinguishability game. $\mathcal{B}$ receives pk_c as it's own challenge, $\mathcal{B}$ embeds pk_c as $\mathsf{pk}_{\mathsf{ch}}$, and proceeds honestly with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate the inner hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{chet\text{-}ind}}(\lambda)$ follows.

Game 3: As Game 2, but instead of adapting $(0, m, \tau, \ell, \widetilde{h}_{1,\ell}, \widetilde{c}_{1,\ell}, \widetilde{r}_{1,\ell}, \mathsf{pk}_{\mathsf{sig}})$ to the new values, directly use CHash.

Transition - Game 2 → Game 3: Assume that the adversary can distinguish this hop. We can then construct an $\mathcal{B}$ which wins the indistinguishability game. $\mathcal{B}$ receives pk'_c as it's own challenge, $\mathcal{B}$ embeds pk'_c as $\mathsf{pk}'_{\mathsf{ch}}$, and proceeds honestly with the exception that it uses the $\mathsf{HashOrAdapt}$ oracle to generate the outer hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{ch\text{-}ind}}(\lambda)$ follows.

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

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

Game 0: The original transparency game, where $b = 0$.

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

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

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

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

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

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

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

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

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

Transition - Game 4 → Game 5: Assume that the adversary can distinguish this hop. We can then construct an $\mathcal{B}$ which wins the indistinguishability game. In particular, the reduction works as follows. $\mathcal{B}$ receives pk_c as it's own challenge, $\mathcal{B}$ embeds pk_c as $\mathsf{pk}_{\mathsf{ch}}$, and proceeds honestly except that it uses the $\mathsf{HashOrAdapt}$ oracle to generate the inner hashes. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_4] - \Pr[S_5]| \leq \nu_{\mathsf{ind\text{-}chet}}(\lambda)$ follows.

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

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 $\mathsf{pk}'_{\mathsf{ch}}$, 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 $\mathcal{B}$. $|\Pr[S_5] - \Pr[S_6]| \leq \nu_{\mathsf{ind\text{-}ch}}(\lambda)$ follows.

We are now in the case $b = 1$, while each hop changes the view of the adversary only negligibly. This concludes the proof.

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

Game 0: The original 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_0.

Game 2: As Game 1, but we now abort, if the adversary was able to find $(\mathsf{pk}^*, \pi^*, m^*, \sigma^*)$ for some message m^* with a τ^* which was never returned by the sanitization oracle. Let this event be E_2.

Transition - Game 1 → Game 2: 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_3 happens, we know by the condition checked in step 4 of Judge that at least one of the tags (either τ^* in σ^*, or τ^π in π^*) 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 sanitizer oracle. If E_3 happens we can output $((0, m^*, \tau^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}^*), r^*_0, (0, m'^*, \tau'^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}_{1,\ell}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}^*), r'^*_0, h^*_0)$, as a valid collision. These values can simply be compiled using π^*, m^*, and σ^*. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{ch\text{-}coll\text{-}res}}(\lambda)$ follows.

Game 3: As Game 2, but we now abort, if the adversary was able to find $(\mathsf{pk}^*, \pi^*, m^*, \sigma^*)$ for a new message m^* which was never returned by the sanitization oracle. Let this event be E_3.

Transition - Game 2 → Game 3: Assume that E_3 happens with non-negligible probability. In the previous games we have already established that the only remaining possibility for the adversary is to re-use tags τ^*, τ^π corresponding to some query/response to the sanitizer oracle. Then, m^* must be fresh, as it was never returned by the sanitization oracle by assumption. Thus, $((0, m^*, \tau^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}^*), r^*_0, (0, m'^*, \tau'^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}_{1,\ell}, \widetilde{r}^*_{1,\ell^*}, \mathsf{pk}^*), r'^*_0, h^*_0)$, is a valid collision. These values can simply be compiled using π^*, m^*, and σ^*. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{ch\text{-}coll\text{-}res}}(\lambda)$ follows.

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

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

Game 0: The original sanitizer-accountability definition.

Game 1: As Game 0, but we abort if the adversary outputs a forgery $(m^*, \sigma^*, \mathsf{pk}^*)$ with $\sigma^* = (\sigma'^*, x^*_0, \widetilde{r}^*_{0,\ell^*}, \widetilde{\tau}^*, \widetilde{c}^*_{1,\ell^*}, \widetilde{h}^*_{0,\ell^*})$, where $(\sigma'^*, (x^*_0, \widetilde{h}^*_{0,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \mathsf{pk}^*_{\mathsf{san}}, \mathsf{pk}^*_{\mathsf{sig}}, \ell^*))$ was never obtained from the signing oracle.

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. Clearly, if $(\sigma'^*, (x_0, \widetilde{h}_{0,\ell}, \widetilde{c}_{1,\ell}, \mathsf{pk}^*, \mathsf{pk}_{\mathsf{sig}}, \ell))$ was never obtained by the challenger, this tuple breaks the strong unforgeability of the underlying signature scheme. The reduction works as follows. We obtain a challenge public key pk_c from a strong unforgeability challenger and embed it as $\mathsf{pk}_{\mathsf{sig}}$. For every required "inner" signature σ', 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 $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\textsf{unf-cma}}(\lambda)$.

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

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

In Game 2 the forgery is different from any query/answer tuple obtained using $\textsf{Sign}$ by definition. Due to the previous hops, the only remaining possibility is a collision in the outer chameleon-hash, i.e., for $h_0^* = h_0'^*$ we have $\textsf{CH.CHashCheck}(\textsf{pk}'^*, (0, m^*, \tau^*, \ell^*, \widetilde{h}^*_{1,\ell^*}, \widetilde{c}^*_{1,\ell^*}, \widetilde{r}^*_{1,\ell^*}, \textsf{pk}_{\textsf{sig}}), r_0^*, h_0^*) = \textsf{CH.CHashCheck}(\textsf{pk}'^*, (0, m'^*, \tau'^*, \ell'^*, \widetilde{h}'^*_{1,\ell'^*}, \widetilde{c}'^*_{1,\ell'^*}, \widetilde{r}'^*_{1,\ell'^*}, \textsf{pk}_{\textsf{sig}}), r_0'^*, h_0'^*) = \texttt{true}$. In this case the $\textsf{Judge}$ algorithm returns $\textsf{San}$ and $\Pr[S_2] = 0$ which concludes the proof.

Invisibility. We prove that our construction is invisible by a sequence of games. The idea is to show that we can simulate the view of the adversary without giving out any useful information at all.

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

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

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

Game 2: As Game 1, but we internally keep all $\textsf{etd}_i$.

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

Game 3: As Game 2, but we encrypt only zeroes instead of the real $\textsf{etd}_i$ in $\textsf{LoRADM}$ independent of whether block are admissible or not. Note, the $\textsf{LoRADM}$ oracle enforces that $\textsf{pk}_{\textsf{san}} = \textsf{pk}'_{\textsf{san}}$. Note, the challenger still knows all $\textsf{etd}_i$, and can thus still sanitize correctly.

Transition - Game 2 → Game 3: A standard reduction, using hybrids, shows that this hop is indistinguishable by the IND-CPA security of the encryption scheme used. $|\Pr[S_2] - \Pr[S_3]| \leq q_h \nu_{\mathsf{ind\text{-}cpa}}(\lambda)$ follows, where q_h is the number of generated ciphertexts by LoRADM.[11]

At this point, the distribution is independent of the LoRADM oracle. Note, the sanitization, and proof oracles, can be still be simulated without any restrictions, as each etd_i is known to the challenger. Thus, the view the adversary receives is now completely independent of the bit b used in the invisibility definition. As each hop only changes the view of the adversary negligibly, our construction is thus proven to be invisible. □

6 Conclusion

We have introduced the notion of chameleon-hashes with ephemeral trapdoors. This primitive allows to prevent the holder of the trapdoor corresponding to the long-term public key from finding collisions. Along with a comprehensive security model we have presented four provably secure constructions. The first one is bootstrapped from any chameleon-hash in a black-box fashion, while the second direct scheme is based on RSA-like assumptions, and the other two on the DL assumption. Applications of this new primitive include, but are not limited to, the first provably secure construction of invisible sanitizable signatures. An interesting open problem is the construction of sanitizable signatures which are invisible and unlinkable at the same time.

Acknowledgements. We are grateful to the anonymous reviewers of PKC 2017 for providing valuable comments and suggestions that helped to significantly improve the presentation of the paper.

References

[ABC$^+$12] J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, a. shelat, and B. Waters. Computing on authenticated data. In *TCC*, pages 1–20, 2012.

[ACdMT05] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS*, pages 159–177, 2005.

[ADK10] S. Alsouri, Ö. Dagdelen, and S. Katzenbeisser. Group-based attestation: Enhancing privacy and management in remote attestation. In *Trust*, pages 63–77, 2010.

[AdM04a] G. Ateniese and B. de Medeiros. Identity-Based Chameleon Hash and Applications. In *Financial Cryptography*, pages 164–180, 2004.

[AdM04b] G. Ateniese and B. de Medeiros. On the key exposure problem in chameleon hashes. In *SCN*, pages 165–179, 2004.

[11] We note that IND-CPA security of the encryption scheme Π is sufficient, as the abort in Game 1 ensures that the adversary can only submit queries with respect to ciphertexts which were previously generated in the reduction, i.e., where we can simply look up the respective values etd_i instead of decryption.

[ADR02] J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *EUROCRYPT 2002*, pages 83–107, 2002.

[AGK08] M. Abe, R. Gennaro, and K. Kurosawa. Tag-kem/dem: A new framework for hybrid encryption. *J. Cryptology*, 21(1):97–130, 2008.

[AMVA16] G. Ateniese, B. Magri, D. Venturi, and E. R. Andrade. Redactable blockchain - or - rewriting history in bitcoin and friends. *IACR Cryptology ePrint Archive*, 2016:757, 2016.

[BBD$^+$10] C. Brzuska, H. Busch, Ö. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In *ACNS*, pages 87–104, 2010.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[BCG07] E. Bresson, D. Catalano, and R. Gennaro. Improved on-line/off-line threshold signatures. In *PKC*, pages 217–232, 2007.

[BDD$^+$11] F. Bao, R. H. Deng, X. Ding, J. Lai, and Y. Zhao. Hierarchical identity-based chameleon hash and its applications. In *ACNS*, pages 201–219, 2011.

[BFF$^+$09] C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of Sanitizable Signatures Revisited. In *PKC*, pages 317–336, 2009.

[BFLS09] C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Sanitizable signatures: How to partially delegate control for authenticated data. In *BIOSIG*, pages 117–128, 2009.

[BFLS10] C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of Sanitizable Signatures. In *PKC*, pages 444–461, 2010.

[BHK15] M. Bellare, D. Hofheinz, and E. Kiltz. Subtleties in the definition of IND-CCA: when and how should challenge decryption be disallowed? *J. Cryptology*, 28(1):29–48, 2015.

[BHPS16] A. Bilzhause, M. Huber, H. C. Pöhls, and K. Samelin. Cryptographically Enforced Four-Eyes Principle. In *ARES*, pages 760–767, 2016.

[BKKP15] O. Blazy, S. A. Kakvi, E. Kiltz, and J. Pan. Tightly-secure signatures from chameleon hash functions. In *PKC*, pages 256–279, 2015.

[BNPS03] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *J. Cryptology*, 16(3):185–215, 2003.

[BPS12] C. Brzuska, H. C. Pöhls, and K. Samelin. Non-Interactive Public Accountability for Sanitizable Signatures. In *EuroPKI*, pages 178–193, 2012.

[BPS13] C. Brzuska, H. C. Pöhls, and K. Samelin. Efficient and Perfectly Unlinkable Sanitizable Signatures without Group Signatures. In *EuroPKI*, pages 12–30, 2013.

[BR14] M. Bellare and T. Ristov. A characterization of chameleon hash functions and new, efficient designs. *J. Cryptology*, 27(4):799–823, 2014.

[BS96] E. Bach and J. O. Shallit. *Algorithmic number theory. Vol. 1. , Efficient algorithms.* Foundations of computing. Cambridge, Mass. MIT Press, 1996. Retirage 1997.

[CJ10] S. Canard and A. Jambert. On extended sanitizable signature schemes. In *CT-RSA*, pages 179–194, 2010.

[CJL12] S. Canard, A. Jambert, and R. Lescuyer. Sanitizable signatures with several signers and sanitizers. In *AFRICACRYPT*, pages 35–52, 2012.

[CL13] S. Canard and R. Lescuyer. Protecting privacy by sanitizing personal data: a new approach to anonymous credentials. In *ASIACCS*, pages 381–392, 2013.

[CLM08] S. Canard, F. Laguillaumie, and M. Milhau. Trapdoor sanitizable signatures and their application to content protection. In *ACNS*, pages 258–276, 2008.

[CRFG08] D. Catalano, M. Di Raimondo, D. Fiore, and R. Gennaro. Off-line/on-line signatures: Theoretical aspects and experimental results. In *PKC*, pages 101–120, 2008.

[CS97] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO*, pages 410–424, 1997.

[CTZD10] X. Chen, H. Tian, F. Zhang, and Y. Ding. Comments and improvements on key-exposure free chameleon hashing based on factoring. In *Inscrypt*, pages 415–426, 2010.

[CZSM07] X. Chen, F. Zhang, W. Susilo, and Y. Mu. Efficient generic on-line/off-line signatures without key exposure. In *ACNS*, pages 18–30, 2007.

[DDH$^+$15] D. Demirel, D. Derler, C. Hanser, H. C. Pöhls, D. Slamanig, and G. Traverso. PRISMACLOUD D4.4: Overview of Functional and Malleable Signature Schemes. Technical report, H2020 Prismacloud, www.prismacloud.eu, 2015.

[DHPS15] D. Derler, C. Hanser, H. C. Pöhls, and D. Slamanig. Towards authenticity and privacy preserving accountable workflows. In *Privacy and Identity Management*, pages 170–186, 2015.

[DHS14] D. Derler, C. Hanser, and D. Slamanig. Blank digital signatures: Optimization and practical experiences. In *Privacy and Identity Management for the Future Internet in the Age of Globalisation*, volume 457, pages 201–215. Springer, 2014.

[dMPPS13] H. de Meer, H. C. Pöhls, J. Posegga, and K. Samelin. Scope of security properties of sanitizable signatures revisited. In *ARES*, pages 188–197, 2013.

[dMPPS14] H. de Meer, H. C. Pöhls, J. Posegga, and K. Samelin. On the relation between redactable and sanitizable signature schemes. In *ESSoS*, pages 113–130, 2014.

[DS15] D. Derler and D. Slamanig. Rethinking privacy for extended sanitizable signatures and a black-box construction of strongly private schemes. In *ProvSec*, pages 455–474, 2015.

[EGM96] S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. *J. Cryptology*, 9(1):35–67, 1996.

[FF15] V. Fehr and M. Fischlin. Sanitizable signcryption: Sanitization over encrypted data (full version). IACR Cryptology ePrint Archive, Report 2015/765, 2015.

[Fis01] M. Fischlin. *Trapdoor Commitment Schemes and Their Applications*. PhD thesis, University of Frankfurt, 2001.

[FKM$^+$16] N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schröder, and M. Simkin. Efficient unlinkable sanitizable signatures from signatures with rerandomizable keys. In *PKC-1*, pages 301–330, 2016.

[GGOT15] E. Ghosh, M. T. Goodrich, O. Ohrimenko, and R. Tamassia. Fully-dynamic verifiable zero-knowledge order queries for network data. *ePrint*, 2015:283, 2015.

[GLW09] W. Gao, F. Li, and X. Wang. Chameleon hash without key exposure based on schnorr signature. *Computer Standards & Interfaces*, 31(2):282–285, 2009.

[GMR88] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, 17:281–308, 1988.

[GOT15] E. Ghosh, O. Ohrimenko, and R. Tamassia. Zero-knowledge authenticated order queries and order statistics on a list. In *ACNS*, volume 2015, pages 149–171, 2015.

[GQZ11] J. Gong, H. Qian, and Y. Zhou. Fully-secure and practical sanitizable signatures. In *Inscrypt*, volume 6584, pages 300–317, 2011.

[Gro06] J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, pages 444–459, 2006.

[GWX07] W. Gao, X. Wang, and D. Xie. Chameleon Hashes Without Key Exposure Based on Factoring. *J. Comput. Sci. Technol.*, 22(1):109–113, 2007.

[HKY15] L. Hanzlik, M. Kutylowski, and M. Yung. Hard invalidation of electronic signatures. In *ISPEC*, pages 421–436, 2015.

[HPS12] F. Höhne, H. C. Pöhls, and K. Samelin. Rechtsfolgen editierbarer signaturen. *Datenschutz und Datensicherheit*, 36(7):485–491, 2012.

[HS13] C. Hanser and D. Slamanig. Blank digital signatures. In *ASIACCS*, pages 95 – 106, 2013.

[HW09] S. Hohenberger and B. Waters. Short and stateless signatures from the RSA assumption. In *CRYPTO*, pages 654–670, 2009.

[KK12] S. A. Kakvi and E. Kiltz. Optimal security proofs for full domain hash, revisited. In *EUROCRYPT*, pages 537–553, 2012.

[KL06] M. Klonowski and A. Lauks. Extended Sanitizable Signatures. In *ICISC*, pages 343–355, 2006.

[KR00] H. Krawczyk and T. Rabin. Chameleon Hashing and Signatures. In *NDSS*, pages 143–154, 2000.

[KSS15] S. Krenn, K. Samelin, and D. Sommer. Stronger security for sanitizable signatures. In *DPM*, pages 100–117, 2015.

[LZCS16] R. W. F. Lai, T. Zhang, S. S. M. Chow, and D. Schröder. Efficient sanitizable signatures without random oracles. In *ESORICS-1*, pages 363–380, 2016.

[Moh10] P. Mohassel. One-time signatures and chameleon hash functions. In *SAC*, pages 302–319, 2010.

[Ped91] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[PPS$^+$13] H. C. Pöhls, S. Peters, K. Samelin, J. Posegga, and H. de Meer. Malleable signatures for resource constrained platforms. In *WISTP*, pages 18–33, 2013.

[PS15] H. C. Pöhls and K. Samelin. Accountable redactable signatures. In *ARES*, pages 60–69, 2015.

[PSP11] H. C. Pöhls, K. Samelin, and J. Posegga. Sanitizable Signatures in XML Signature - Performance, Mixing Properties, and Revisiting the Property of Transparency. In *ACNS*, volume 6715 of *LNCS*, pages 166–182. Springer, 2011.

[RMS07] Q. Ren, Y. Mu, and W. Susilo. Mitigating Phishing by a New ID-based Chameleon Hash without Key Exposure. In *AusCERT*, pages 1–13, 2007.

[ST01] A. Shamir and Y. Tauman. Improved online/offline signature schemes. In *CRYPTO*, pages 355–367, 2001.

[YSL10] D. H. Yum, J. W. Seo, and P. J. Lee. Trapdoor sanitizable signatures made easy. In *ACNS*, pages 53–68, 2010.

[Zha07] R. Zhang. Tweaking TBE/IBE to PKE transforms with Chameleon Hash Functions. In *ACNS*, pages 323–339, 2007.

```
Experiment $\mathsf{IND\text{-}T}^{\Pi}_{\mathcal{A}}(\lambda)$
  $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}}) \leftarrow \mathsf{KGen}_{\mathsf{enc}}(1^{\lambda})$
  $b \leftarrow \{0,1\}$
  $(m_0, m_1, \mathtt{state}_{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk}_{\mathsf{enc}})$
    where $\mathcal{O} \leftarrow \mathsf{Dec}(\mathsf{sk}_{\mathsf{enc}}, \cdot)$ if $T = CCA2$ and $\mathcal{O} \leftarrow \emptyset$ otherwise.
  if $m_0 \notin \mathcal{M} \ \vee \ m_1 \notin \mathcal{M}$, let $c \leftarrow \bot$
  else, let $c \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{enc}}, m_b)$
  $a \leftarrow \mathcal{A}^{\mathcal{O}}(c, \mathtt{state}_{\mathcal{A}})$
    where $\mathcal{O} \leftarrow \mathsf{Dec}'(\mathsf{sk}_{\mathsf{enc}}, \cdot)$ if $T = CCA2$ and $\mathcal{O} \leftarrow \emptyset$ otherwise.
    $\mathsf{Dec}'(\mathsf{sk}_{\mathsf{enc}}, \cdot)$ behaves as $\mathsf{Dec}$, but returns $\bot$ if queried with $c$.
  return 1, if $a = b$
  return 0
```

Fig. 9. IND-T security with $T \in \{CPA, CCA2\}$.

[ZSnS03] F. Zhang, R. Safavi-naini, and W. Susilo. Id-based chameleon hashes from bilinear pairings. *IACR Cryptology ePrint Archive*, 2003:208, 2003.

A Security Definitions Building Blocks

A.1 Public-Key Encryption Schemes Π

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

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

Definition 18 (IND-T Security). *An encryption scheme Π is* IND-T secure *with $T \in \{CPA, CCA2\}$, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that $\left|\Pr[\mathsf{IND\text{-}T}^{\Pi}_{\mathcal{A}}(1^{\lambda}) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$. The corresponding experiment is depicted in Fig. 9.*

A.2 Pseudo-Random Functions PRF

Definition 19 (Pseudo-Random Functions (PRFs)). *A pseudo-random function* PRF *consists of two algorithms* $(\mathsf{KGen}_{\mathsf{prf}}, \mathsf{Eval}_{\mathsf{prf}})$ *such that:*

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

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

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

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

Fig. 11. Pseudo-Randomness

Pseudo-Randomness. We require that PRF is actually pseudo-random. 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 20 (Pseudo-Randomness). *A pseudo-random function* PRF *is* pseudo-random, *if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that $\left|\Pr[\mathsf{Pseudo\text{-}Randomness}^{\mathsf{PRF}}_{\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$. The corresponding experiment is depicted in Fig. 10.*

A.3 Pseudo-Random Generators PRG

We assume PRGs with a constant stretching factor of 2 below, as this is sufficient for our setting.

Definition 21 (Pseudo-Random Number-Generators). *A pseudo-random number-generator* PRG *consists of one algorithm* ($\mathsf{Eval}_{\mathsf{prg}}$) *such that:*

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

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

Definition 22 (Pseudo-Randomness). *A pseudo-random number-generator* PRG *is* pseudo-random, *if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that $\left|\Pr[\mathsf{Pseudo\text{-}Randomness}^{\mathsf{PRG}}_{\mathcal{A}}(1^\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$. The corresponding experiment is depicted in Fig. 11.*

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

Fig. 12. Strong Unforgeability

A.4 Digital Signatures Σ

Definition 23 (Digital Signatures). *A signature scheme Σ is a triple* $(\mathsf{KGen_{sig}}, \mathsf{Sign}, \mathsf{Verify})$ *of ppt algorithms such that:*

$\mathsf{KGen_{sig}}$. *The algorithm* $\mathsf{KGen_{sig}}$ *on input security parameter λ outputs the public and corresponding private key:* $(\mathsf{sk_s}, \mathsf{pk_s}) \leftarrow \mathsf{KGen_{sig}}(1^\lambda)$*.*

Sign. *The algorithm* Sign *gets as input the secret key* $\mathsf{sk_s}$*, and the message $m \in \mathcal{M}$ and outputs a signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk_s}, m)$.*

Verify. *The algorithm* Verify *receives as input a public key* $\mathsf{pk_s}$ *a message m and a signatures σ and outputs a decision bit $d \in \{\mathtt{false}, \mathtt{true}\}$: $d \leftarrow \mathsf{Verify}(\mathsf{pk_s}, m, \sigma)$.*

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 $(\mathsf{sk_s}, \mathsf{pk_s}) \leftarrow \mathsf{KGen_{sig}}(1^\lambda)$, for all $m \in \mathcal{M}$ we have $\mathsf{Verify}(\mathsf{pk_s}, m, \mathsf{Sign}(\mathsf{sk_s}, m)) = \mathtt{true}$. This definition captures perfect correctness.

Strong Unforgeability. Now, we define strong unforgeability of digital signature schemes, as given by An et al. [ADR02]. In a nutshell, we require that an adversary $\mathcal{A}$ cannot (except with negligible probability) come up with *any* new valid signature σ^* for a message m^*. Moreover, the adversary $\mathcal{A}$ can adaptively query for new signatures.

Definition 24 (Strong Unforgeability). *A signature scheme Σ is* strongly unforgeable*, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{seUNF\text{-}CMA}^{\Sigma}_{\mathcal{A}}(1^\lambda) = 1] \leq \nu(\lambda)$*. The corresponding experiment is depicted in Fig. 12.*

Definition 25 (Secure Digital Signatures). *We call a signature scheme Σ secure, if it is correct, and strongly unforgeable.*

A concrete instantiation satisfying Definition 25 is RSA-FDH, where the signer also proves the well-formedness of the public key, i.e., that it defines a permutation and is not lossy [KK12]. This can, e.g., be achieved by requiring that the public exponent e is prime and greater than the modulus n, while a verifier also has to check that $\sigma \in \mathbb{Z}_n^*$. See also Lemma 1 in App. F.2.

```
**Experiment** $\mathsf{Zero\text{-}Knowledge}_{\mathcal{A}}^{\mathrm{NIZKPoK}}(\lambda)$
  $b \leftarrow \{0,1\}$
  $(\mathsf{crs}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$
  $a \leftarrow \mathcal{A}^{\mathsf{P}_b(\cdot,\cdot)}(\mathsf{crs})$
    where oracle $\mathsf{P}_0$ on input $(x, w)$:
      return $\pi \leftarrow \{(w) : R(x, w) = 1\}$, if $(x, w) \in L$
      return $\perp$
    and oracle $\mathsf{P}_1$ on input $(x, w)$:
      return $\pi \leftarrow \mathcal{S}_2(\mathsf{crs}, \tau, x)$, if $(x, w) \in L$
      return $\perp$
  return 1, if $a = b$
  return 0
```

Fig. 13. Zero-Knowledge

A.5 Non-Interactive Proof Systems NIZKPoK

We now provide the formal definitions for the NIZKPoKs we need, derived from Groth [Gro06].

Definition 26 (Completeness). *A non-interactive proof system is complete, if for all $\lambda \in \mathbb{N}$, for all "suitable" L, for all $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, L)$, for all $x \in L$, for all w such that $R(x, w) = 1$, for all $\pi \leftarrow \mathrm{NIZKPoK}\{(w) : R(x, w)\}$, we have that $\mathsf{Verify}(x, \pi) = \mathtt{true}$.*

This captures perfect completeness.

Definition 27 (Zero-Knowledge). *A non-interactive proof system is zero-knowledge, if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\left| \Pr[\mathsf{Zero\text{-}Knowledge}_{\mathcal{A}}^{\mathrm{NIZKPoK}}(\lambda) = 1] - \frac{1}{2} \right| \leq \nu(\lambda)\,.$$

The corresponding experiment is depicted in Fig. 13.

Definition 28 (Simulation-Sound Extractability). *A zero-knowledge non-interactive proof system is simulation-sound extractable, if there exists an efficient extractor $\mathsf{E} = (\mathcal{S}, \mathcal{E})$ such that it holds that $(\mathsf{crs}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$ is identically distributed as $(\mathsf{crs}, \tau, \xi) \leftarrow \mathcal{E}(1^\lambda)$ when restricted to (crs, τ) and such that it holds for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that $\Pr[\mathsf{SimSoundExt}_{\mathcal{A},\mathsf{E}}^{\mathrm{NIZKPoK}}(\lambda) = 1] \leq \nu(\lambda)$. The corresponding experiment is depicted in Fig. 14.*

B Additional Security Definitions for SSS

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

```
**Experiment** $\mathsf{SimSoundExt}^{\mathrm{NIZKPoK}}_{\mathcal{A},\mathsf{E}}(\lambda)$
  $(\mathsf{crs}, \tau, \xi) \leftarrow \mathcal{S}(1^\lambda)$
  $(x, \pi) \leftarrow \mathcal{A}^{\mathsf{Sim}(\cdot)}(\mathsf{crs})$
    where oracle $\mathsf{Sim}$ on input $x$:
      obtain $\pi \leftarrow \mathcal{S}_2(\mathsf{crs}, \tau, x)$
      set $\mathcal{Q}^{\mathsf{Sim}} \leftarrow \mathcal{Q}^{\mathsf{Sim}} \cup \{(x, \pi)\}$
  $w \leftarrow \mathcal{E}(\mathsf{crs}, \xi, x, \pi)$
  return 1, if $\mathsf{Verify}(x, \pi) = \mathtt{true} \ \wedge \ (x, w) \notin R \ \wedge \ (x, \pi) \notin \mathcal{Q}^{\mathsf{Sim}}$
  return 0
```

Fig. 14. Simulation Sound Extractability

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

Fig. 15. Unforgeability

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

Fig. 16. Immutability

Definition 29 (Unforgeability). *An* SSS *is unforgeable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Unforgeability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Figure 15.*

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

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

Fig. 17. Privacy

Definition 30 (Immutability). *An* SSS *is immutable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Immutability}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Figure 16.*

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. Afterwards, the adversary has to decide which message (left or right) was used to generate the sanitized one. The adversary receives full adaptive oracle access.

Definition 31 (Privacy). *An* SSS *is private, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\left|\Pr[\mathsf{Privacy}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$, *where the corresponding experiment is defined in Figure 17.*

Transparency. Transparency guarantees that the accountable party of a message m remains anonymous. This is important if discrimination may follow [ACdMT05, BFF$^+$09]. In a nutshell, the adversary has to decide whether it sees a freshly computed signature, or a sanitized one. The adversary has full (but proof-restricted) adaptive oracle access. We require the proof-restriction to avoid trivial attacks. Moreover, we have adjusted the definition to account for some subtleties regarding the restrictions of the proof oracle, in the sense of Bellare et al. for IND-CCA2 security [BHK15].

Definition 32 ((Proof-Restricted) Transparency). *An* SSS *is proof-restricted transparent, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\left|\Pr[\mathsf{Transparency}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$, *where the corresponding experiment is defined in Figure 18.*

In this paper, we use "transparency", even if we mean proof-restricted transparency. Moreover, our construction actually achieves the stronger notion of transparency, with the same arguments given by Brzuska et al. [BFLS10]. However, the proof-restricted version seems to be more natural.

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

Fig. 18. (Proof-Restricted) Transparency

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

Fig. 19. Signer Accountability

Signer-Accountability. For signer-accountability, a signer should not be able to blame a sanitizer if the sanitizer is actually not responsible for a given message/signature pair never issued by the sanitizer. 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. Note, this definition does not take the signature into account.

Definition 33 (Signer-Accountability). *An* SSS *is signer-accountable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Sig\text{-}Accountability}_{\mathcal{A}}^{\mathsf{SSS}}(\lambda) = 1] \leq \nu(\lambda)$, *where the experiment is defined in Figure 19.*

Sanitizer-Accountability. Sanitizer-accountability requires that the sanitizer cannot blame the signer for a message/signature pair not created by the signer. In particular, the adversary has to make Proof generate a proof π which makes

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

Fig. 20. Sanitizer Accountability

```
$\textbf{Experiment}\ \mathsf{Unlinkability}_{\mathcal{A}}^{\mathsf{SSS}}(\lambda)$
    $\mathsf{pp_{sss}} \leftarrow \mathsf{SSSParGen}(1^\lambda)$
    $(\mathsf{sk_{san}}, \mathsf{pk_{san}}) \leftarrow \mathsf{KGen_{san}}(\mathsf{pp_{sss}})$
    $b \leftarrow \{0, 1\}$
    $a \leftarrow \mathcal{A}^{\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk_{san}}), \mathsf{LoRSanit}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{sk_{san}},b)}(\mathsf{pk_{san}})$
        where oracle $\mathsf{LoRSanit}$ on input of $m_0, \mathrm{MOD}_0, \sigma_0, m_1, \mathrm{MOD}_1, \sigma_1, \mathsf{pk_{sig}}, b$:
            return $\perp$, if $\mathrm{ADM}_0 \neq \mathrm{ADM}_1 \lor \mathrm{MOD}_0(m_0) \neq \mathrm{MOD}_1(m_1) \lor$
                $\mathrm{MOD}_0(\mathrm{ADM}_0) \neq \mathrm{MOD}_1(\mathrm{ADM}_1) \lor$
            $\mathsf{Verify}(m_0, \sigma_0, \mathsf{pk_{sig}}, \mathsf{pk_{san}}) \neq \mathsf{Verify}(m_1, \sigma_1, \mathsf{pk_{sig}}, \mathsf{pk_{san}})$
            return $(m', \sigma') \leftarrow \mathsf{Sanit}(m_b, \mathrm{MOD}_b, \sigma_b, \mathsf{pk_{sig}}, \mathsf{sk_{san}})$
    return 1, if $a = b$
    return 0
```

Fig. 21. Unlinkability

Judge decide that for a given (m^*, σ^*) generated by $\mathcal{A}$ the signer is accountable, while it is not. Thus, the adversary $\mathcal{A}$ gains access to all signer-related oracles.

Definition 34 (Sanitizer-Accountability). *An* SSS *is sanitizer-accountable, if for any ppt adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{San\text{-}Accountability}_{\mathcal{A}}^{\mathsf{SSS}}(\lambda) = 1] \leq \nu(\lambda)$, *where the experiment is defined in Figure 20.*

We do not consider unlinkability [BFLS10, BPS13, FKM$^+$16, LZCS16] in our construction, as it seems to be very hard to achieve with the underlying construction paradigm, but provide the definition for the sake of completeness.

Unlinkability. Unlinkability prohibits an adversary to decide how a signature was generated, i.e., from which signature a sanitized signature was derived. This is the stronger definition by Brzuska et al. [BPS13], where even the signer can be malicious. This game is similar to privacy with the same constraints. But in contrast to the privacy game, the adversary can also input the signatures to be used. It receives full oracle access, while the signing, and the proof oracles, can be simulated by the adversary.

```
**Experiment** $\mathsf{Pubaccountability}_{\mathcal{A}}^{\mathsf{SSS}}(\lambda)$
  $\mathsf{pp_{sss}} \leftarrow \mathsf{SSSParGen}(1^\lambda)$
  $(\mathsf{sk_{sig}}, \mathsf{pk_{sig}}) \leftarrow \mathsf{KGen_{sig}}(\mathsf{pp_{sss}})$
  $(\mathsf{sk_{san}}, \mathsf{pk_{san}}) \leftarrow \mathsf{KGen_{san}}(\mathsf{pp_{sss}})$
  $(\mathsf{pk}^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\mathsf{sk_{sig}},\cdot,\cdot),\mathsf{Sanit}(\cdot,\cdot,\cdot,\cdot,\mathsf{sk_{san}})}(\mathsf{pk_{sig}}, \mathsf{pk_{san}})$
    for $i = 1, 2, \ldots, q$ let $(m_i, \mathrm{ADM}_i, \mathsf{pk}_{\mathsf{san},i})$ and $\sigma_i$
      index the queries/answers to/from $\mathsf{Sign}$
    for $j = 1, 2, \ldots, q'$ let $(m_j, \mathrm{MOD}_j, \sigma_j, \mathsf{pk}_{\mathsf{sig},j})$ and $(m'_j, \sigma'_j)$
      index the queries/answers to/from $\mathsf{Sanit}$
  return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk_{sig}}, \mathsf{pk}^*) = \mathtt{true} \land$
    $\forall i \in \{1, 2, \ldots, q\} : (\mathsf{pk}^*, m^*, \sigma^*) \neq (\mathsf{pk}_{\mathsf{san},i}, m_i, \sigma_i) \land$
    $\mathsf{Judge}(m^*, \sigma^*, \mathsf{pk_{sig}}, \mathsf{pk}^*, \bot) = \mathsf{Sig}$
  return 1, if $\mathsf{Verify}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk_{san}}) = \mathtt{true} \land$
    $\forall j \in \{1, 2, \ldots, q'\} : (\mathsf{pk}^*, m^*, \sigma^*) \neq (\mathsf{pk}_{\mathsf{sig},j}, m'_j, \sigma'_j) \land$
    $\mathsf{Judge}(m^*, \sigma^*, \mathsf{pk}^*, \mathsf{pk_{san}}, \bot) = \mathsf{San}$
  return 0
```

Fig. 22. Public Accountability

Definition 35 (Unlinkability). *An* SSS *is unlinkable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that* $\left|\Pr[\mathsf{Unlinkability}_{\mathcal{A}}^{\mathsf{SSS}}(\lambda) = 1] - \frac{1}{2}\right| \leq \nu(\lambda)$, *where the corresponding experiment is defined in Figure 21.*

Non-Interactive Public Accountability. Non-interactive public accountability allows everyone to decide whether a sanitizer was involved. This is modeled by requiring that Judge works with an empty proof, i.e., $\pi = \bot$. Hence, no secret keys are required to find the accountable party.

Definition 36 (Non-Interactive Public Accountability). *An* SSS *is non-interactive publicly accountable, if for any efficient adversary $\mathcal{A}$ there exists a negligible function ν such that* $\Pr[\mathsf{Pubaccountability}_{\mathcal{A}}^{\mathsf{SSS}}(\lambda) = 1] \leq \nu(\lambda)$, *where the corresponding experiment is defined in Figure 22.*

C Additional Assumptions

RSA Problems and RSA Assumptions. Let $(n, p, q, e, d) \leftarrow \mathsf{RSAKGen}(1^\lambda)$ be an instance generator which returns an RSA modulus $n = pq$, where p and q are distinct primes, $e > 1$ an integer co-prime to $\varphi(n)$, and $de \equiv 1 \bmod \varphi(n)$. We require that RSAKGen always outputs moduli with the same bit-length, based on λ. The RSA problem associated to RSAKGen is, given n, e and $y \leftarrow \mathbb{Z}_n^*$ to find an x s.t. $x^e \equiv y \bmod n$. Likewise, the RSA assumption associated to RSAKGen now states that for every efficient adversary $\mathcal{A}$, $\Pr[(n, p, q, e, d) \leftarrow \mathsf{RSAKGen}(1^\lambda), y \leftarrow \mathbb{Z}_n^*, x \leftarrow \mathcal{A}(n, e, y) : x^e \equiv y \bmod n] \leq \nu(\lambda)$ for some negligible function ν.

The one-more RSA assumption associated to RSAKGen 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 returns a random element $y_i \in \mathbb{Z}_n^*$, to, given n and e, invert more elements received by the challenge oracle than calls to the inversion oracle. The corresponding assumption formally states that for every efficient adversary $\mathcal{A}$ $\Pr[(n, p, q, e, d) \leftarrow \mathsf{RSAKGen}(1^\lambda), X \leftarrow \mathcal{A}(n, e)^{\mathcal{C}(n), \mathcal{I}(d,n,\cdot)}$: more values returned by $\mathcal{C}$ are inverted than queries to $\mathcal{I}] \leq \nu(\lambda)$ for some negligible function ν [BNPS03].

We sometimes require that e is larger than any possible n w.r.t. λ (or even for $e > n^3$), and that it is prime. We write $e > n$ as a shorthand notation. Re-stating the assumptions with these conditions is straightforward. In this case, we also require 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 requiring 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. We leave this to the concrete instantiation of $\mathsf{RSAKGen}$.

D Chameleon-Hashes Revisited

We first restate the construction by Brzuska et al. [BFF$^+$09] which involves a tag τ, and is secure under the standard RSA assumption in the random oracle model. Note, one can even see τ as part of the randomness. We fit the presentation into our framework

Construction 4 (Chameleon-Hash) *Let $\mathcal{H}_n : \{0,1\}^* \to \mathbb{Z}_n^*$, $n \in \mathbb{N}$, denote a random oracle. Now, let* $\mathsf{CH} := (\mathsf{CParGen}, \mathsf{CKGen}, \mathsf{CHash}, \mathsf{CHashCheck}, \mathsf{Adapt})$ *such that:*

$\mathsf{CParGen}$. *The algorithm* $\mathsf{CParGen}$ *generates the public parameters in the following way:*

1. *Return $\emptyset$.*

CKGen. *The algorithm* CKGen *generates the key pair in the following way:*

1. *Generate two primes p and q using $\mathsf{RSAKGen}(1^\lambda)$, which also outputs e co-prime to $\varphi(n)$. Set $\mathsf{sk_{ch}} \leftarrow (p, q)$. Let $n \leftarrow pq$. Set $\mathsf{pk_{ch}} \leftarrow (n, e)$.*
2. *Return $(\mathsf{pk_{ch}}, \mathsf{sk_{ch}})$.*

CHash. *To hash a message m, along with tag τ, w.r.t. $\mathsf{pk_{ch}} = (n, e)$ do:*

1. *Draw $r \leftarrow \mathbb{Z}_n^*$.*
2. *Let $g \leftarrow \mathcal{H}_n(\tau, m)$, and $h \leftarrow gr^e \bmod n$.*
3. *Return (h, r).*

$\mathsf{CHashCheck}$. *To check a hash h' w.r.t. a message m, tag τ, $\mathsf{pk_{ch}} = (n, e)$, and randomness r do:*

1. *Let $g \leftarrow \mathcal{H}_n(\tau, m)$, and $h \leftarrow gr^e \bmod n$.*
2. *Return* `true`*, if $h = h'$, and* `false` *otherwise.*

Adapt. *To find a collision w.r.t. m, m', tag τ, randomness r, hash h, and $\mathsf{sk_{ch}}$ do:*

1. *Compute $g \leftarrow \mathcal{H}_n(\tau, m)$, and $h \leftarrow gr^e \bmod n$.*
2. *Draw $\tau' \leftarrow \{0,1\}^\lambda$.*
3. *Compute $g' \leftarrow \mathcal{H}_n(\tau', m')$ and $r' \leftarrow (h(g'^{-1}))^d \bmod n$.*
4. *Return r'.*

D.1 Our Modified Construction

We now present the modified construction, inspired by the ideas given by Brzuska et al. [BFF+09]. It should be clear that indistinguishability still holds in this setting, except that the modified construction achieves stronger collision-resistance under a different assumption, and is unique, which we prove on its own.

Construction 5 (Modified Chameleon-Hash) *Let* $\mathcal{H}_n : \{0,1\}^* \to \mathbb{Z}_n^*$, $n \in \mathbb{N}$, *denote a random oracle. Now, let* $\mathsf{CH} := (\mathsf{CParGen}, \mathsf{CKGen}, \mathsf{CHash}, \mathsf{CHashCheck}, \mathsf{Adapt})$ *such that:*

$\mathsf{CParGen}$. *The algorithm* $\mathsf{CParGen}$ *generates the public parameters in the following way:*

1. *Call* $\mathsf{RSAKGen}$ *with the restriction* $e > n$, *and* e *prime. Return* e.

CKGen. *The algorithm* CKGen *generates the key pair in the following way:*

1. *Generate* p, q *using* $\mathsf{RSAKGen}(1^\lambda)$.
2. *Let* $n = pq$.
3. *Compute* d *such that* $ed \equiv 1 \bmod \varphi(n)$.
4. *Return* $(\mathsf{pk_{ch}}, \mathsf{sk_{ch}}) = (n, d)$.

CHash. *To hash a message* m *w.r.t.* $\mathsf{pk_{ch}}$ *do:*

1. *Draw* $r \leftarrow \mathbb{Z}_n^*$.
2. *Let* $h \leftarrow \mathcal{H}_n(m)r^e \bmod n$.
3. *Return* (h, r).

$\mathsf{CHashCheck}$. *To check a hash* h' *w.r.t. a message* m, *randomness* r, *and* $\mathsf{pk_{ch}}$ *do:*

1. *If* $r \notin \mathbb{Z}_n^*$, *return* `false`.
2. *Let* $h \leftarrow \mathcal{H}_n(m)r^e \bmod n$.
3. *Return* `true`, *if* $h = h'$, *and* `false` *otherwise.*

Adapt. *To find a collision w.r.t.* m, m', *randomness* r, *hash* h, *and* $\mathsf{sk_{ch}}$ *do:*

1. *If* $\mathsf{CHashCheck}(\mathsf{pk_{ch}}, m, r, h) = $ `false`, *return* $\perp$.
2. *If* $m = m'$, *return* r.
3. *Let* $g \leftarrow \mathcal{H}_n(m)$, *and* $y \leftarrow gr^e \bmod n$.
4. *Let* $g' \leftarrow \mathcal{H}_n(m')$.
5. *Return* $r' \leftarrow (y(g'^{-1}))^d \bmod n$.

Theorem 4. *If the one-more RSA-inversion assumption holds, then the above construction is secure in the random oracle model.*

Proof. We prove each property separately.

Indistinguishability. Indistinguishability is straight forward to see; the randomness r is freshly drawn in the challenge oracle and RSA defines a permutation. The proof is therefore omitted.

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

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

Game 1: As Game 0, but instead of using the e from the system parameters, we embed the e received from a one-more RSA challenger as $\mathsf{pp}_{\mathsf{ch}}$. 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 $(m^*, r^*, m'^*, r'^*, h^*)$. 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 $\mathsf{pk}_{\mathsf{ch}}$. 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, $\mathcal{B}$ asks its challenge oracle $\mathcal{C}$ to provide a challenge $c_i \in \mathbb{Z}_n^*$. This challenge is embedded as the response to m_i. It stores $(m_i, c_i, \perp)$ in an internal table. This does not change the view of the adversary either. However, it remains open how to simulate the adaption oracle. Assume, for now, that m_0 is supposed to be adapted to m_1. If $m_0 = m_1$, proceed as in the algorithm. If there is no tuple (m_0, c_0, z_0), with $z_0 \neq \perp$, query the inversion oracle with c_0 to receive z_0. Update record $(m_0, c_0, \perp)$ to (m_0, c_0, z_0). If there is no tuple (m_1, c_1, z_1), with $z_1 \neq \perp$, query the inversion oracle with c_1 to receive z_1. Update record $(m_1, c_1, \perp)$ to (m_1, c_1, z_1). Return $r' \leftarrow rz_0(z_1)^{-1} \bmod n$. Then, at some point in time, the adversary returns (m, r, m', r', h). We then know (by construction) that $\mathcal{H}_n(m)r^e \equiv \mathcal{H}_n(m')r'^e \bmod n$. If there is no record for m *and* no record for m', query the inversion oracle for the root z of $\mathcal{H}(m)$, and update record $(m, c, \perp)$ to (m, c, z). Then, by definition of the security game, we know that m' is fresh, and there exists a record $(m', c', \perp)$. We can then extract $\mathcal{H}_n(m')^d = z'$ by calculating $\mathcal{H}(m')^d \equiv r'^{-1}zr \bmod n$. 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, c_i, z_i), $z_i \neq \perp$ exists, along with $(c', \mathcal{H}_n(m')^d)$. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{om\text{-}rsa}}(\lambda)$ follows.

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

Uniqueness. We now prove that the above construction is unique using a sequence of games:

Game 0: The original uniqueness game.

Game 1: As game 0, but the challenger aborts, if the adversary finds randomness $r^* \neq r'^*$, a public key $\mathsf{pk}^* = n$, a message m^*, and a hash h^* such that $\mathsf{CHashCheck}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{CHashCheck}(\mathsf{pk}^*, m^*, r'^*, h^*) = \mathtt{true}$. Let this abort event be denoted E_1.

Transition - Game 0 → Game 1: This cannot happen, as RSA (with the given restrictions on e and r) defines a permutation, and random oracles behave as functions, regardless of the choice of n. $|\Pr[S_0] - \Pr[S_1]| = 0$ follows. See also Lemma 1.

This proves that our construction is unique.

□

E Two Additional Direct Constructions

E.1 A Direct Construction From RSA-Like Assumptions

We extend the RSA-based chameleon-hash given in App. D.1 (which is itself based on the construction given by Brzuska et al. [BFF+09], see App. D) using the technique used for accumulators by Pöhls et al. [PPS+13]. In our construction, the trapdoor is an additional RSA-modulus n'. Only if the factorization of $n' = p'q'$, contained in etd, 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. We note that the condition $e > n^3$ implies that $e > nn'$, and thus $\gcd(\varphi(nn'), e) = 1$, which makes our analysis simpler (cf. Lemma 1 in App. F.2).

We do not explicitly check this in the algorithms.

Construction 6 **(CHET from RSA-like Assumptions)** *Let $\mathcal{H}_n : \{0,1\}^* \to \mathbb{Z}_n^*$, $n \in \mathbb{N}$, denote a random oracle. Let* CHET *be defined as:*

CParGen. *The algorithm* CParGen *generates the public parameters in the following way:*

1. *Call* RSAKGen *with the restriction $e > n^3$, and e prime. Return e.*

CKGen. *The algorithm* CKGen *generates the key pair in the following way:*

1. *Generate two primes p and q using* $\mathsf{RSAKGen}(1^\lambda)$. *Set $\mathsf{sk_{ch}} \leftarrow (p, q)$. Let $n \leftarrow pq$. Set $\mathsf{pk_{ch}} \leftarrow n$.*
2. *Return $(\mathsf{sk_{ch}}, \mathsf{pk_{ch}})$.*

CHash. *To hash a message m w.r.t. $\mathsf{pk_{ch}} = n$ do:*

1. *Generate two primes p' and q' using* $\mathsf{RSAKGen}(1^\lambda)$. *Set $\mathsf{etd} \leftarrow (p', q')$, and $n' \leftarrow p'q'$.*
2. *If $\gcd(n, n') \neq 1$, go to 1.*
3. *Draw $r \leftarrow \mathbb{Z}_{nn'}^*$.*
4. *Let $g \leftarrow \mathcal{H}_{nn'}(m)$, and $h \leftarrow gr^e \bmod nn'$.*
5. *Return $((h, n'), r, \mathsf{etd})$.*

CHashCheck. *To check whether a given hash h is valid on input $\mathsf{pk_{ch}} = n$, m, and r, do:*

```
    1. Return $\bot$, if $r \notin \mathbb{Z}^*_{nn'}$.
    2. Let $g \leftarrow \mathcal{H}_{nn'}(m)$, and $h' \leftarrow gr^e \bmod nn'$.
    3. Return true, if $h = (h', n')$.
    4. Return false.
Adapt. To find a collision w.r.t. $m$, $m'$, randomness $r$, hash $h$, trapdoor information etd, and $\mathsf{sk}_{\mathsf{ch}}$ do:
    1. Check that $n' = p'q'$, where $p'$ and $q'$ is taken from etd. If this is not the case, return $\bot$.
    2. If $\mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, m, r, h) = \mathtt{false}$, return $\bot$.
    3. Compute $d$ s.t. $de \equiv 1 \bmod \varphi(nn')$.
    4. Let $g \leftarrow \mathcal{H}_{nn'}(m)$, and $h \leftarrow gr^e \bmod nn'$.
    5. Let $g' \leftarrow \mathcal{H}_{nn'}(m')$ and $r' \leftarrow (h(g'^{-1}))^d \bmod nn'$.
    6. Return $r'$.
```

Theorem 5. *If the one-more RSA-inversion assumption holds, then the above construction is secure in the random-oracle model.*

Proof. We need to prove that our construction is indistinguishable, unique, 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. See also Lemma 1.

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 $\mathsf{pp}_{\mathsf{ch}}$. 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 $(m^*, r^*, m'^*, r'^*, h^*)$. 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 $\mathsf{pk}_{\mathsf{ch}}$ 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 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.[12] 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. Then, at some point in time, the adversary returns $(m^*, r^*, m'^*, r'^*, h^*)$. 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 $\mathcal{H}_{nn'}(m^*)$, and update record $(m^*, n', c^*, r^*, \perp)$ to (m^*, n', c^*, u^*, z^*). Then, by definition of the security game, we know that m'^* is fresh, 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 $(c'^*, z'^* \bmod n)$. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{om\text{-}rsa}}(\lambda)$ 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 $\mathsf{pp_{ch}}$. 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.

[12] Lemma 2 directly provides a polynomial bound on the expected number of needed samples. To have a strictly polynomial-time $\mathcal{B}$, we abort if we need $\mathsf{p}(\lambda)$ times more samples than expected, for some fixed polynomial p. Clearly, this only happens with at most negligible probability, and therefore does not significantly change the winning probability in the following.

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 $(m^*, r^*, m'^*, r'^*, h^*)$ which breaks the private collision-resistance of our construction. Let this event be E_3.

Transition - Game 2 → Game 3: First, note that without loss of generality we only need to consider such adversaries $\mathcal{A}$ that only make a single call to the CHash' oracle, as it can simulate all other calls (except for the h^*) internally.[13] Assume now that $\Pr[E_3]$ is non-negligible. We can then construct an adversary $\mathcal{B}$ which breaks the one-more RSA assumption with non-negligible probability. $\mathcal{B}$ simulates the CHash' oracle by embedding the modulus it received from its own RSA challenger as n'.[14] For computing $\mathcal{H}_{nn'}(m)$, it asks its own challenger for challenges in $\mathbb{Z}^*_{n'}$ until it receives a challenge c that also lies in $\mathbb{Z}^*_{nn'}$ (as before, this happens after a polynomial number of steps by Lemma 2). It then chooses a random $u \leftarrow \mathbb{Z}^*_{nn'}$, sets $\mathcal{H}_{nn'}(m) = cu^e \bmod nn'$, and computes its response as in the original algorithm, i.e., it outputs $((\mathcal{H}_{nn'}(m)r^e \bmod nn', n'), r)$ for a random r. All other queries to the $\mathcal{H}_{nn'}(m_j)$ oracle are replied by v_j^e for a fresh $v_j \leftarrow \mathbb{Z}^*_{nn'}$, and the pairs (m_j, v_j) are stored internally. As before, we assume that $\mathcal{A}$ did all the random oracle queries before it outputs its messages (otherwise, $\mathcal{B}$ makes the necessary queries). Eventually, $\mathcal{A}$ outputs $(m^*, r^*, m'^*, r'^*, h^*)$ with $\mathsf{CHashCheck}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{CHashCheck}(\mathsf{pk}^*, m'^*, r'^*, h^*) = \mathtt{true}$ with h^* as returned by $\mathcal{B}$ and $m'^* \neq m^*$. $\mathcal{B}$ then looks up v^* such that $\mathcal{H}_{nn'}(m^*) = v^{*e}$. By assumption it then holds that $\mathcal{H}_{nn'}(m^*)r^{*e} = \mathcal{H}_{nn'}(m'^*)r'^{*e}$, i.e., that $v^{*e}r^{*e} = cu^e r^e \bmod nn'$, or equivalently that $c = \left(v^* r^* u^{-1} r^{-1}\right)^e \bmod nn'$. $\mathcal{B}$ now outputs $(c, x = v^* r^* u^{-1} r^{-1} \bmod n')$ and returns it to its one-more RSA challenger. It is easy to see that $c = x^e \bmod n'$ holds: $c = \left(v^* r^* u^{-1} r^{-1}\right)^e \bmod nn'$ means that $c = \left(v^* r^* u^{-1} r^{-1}\right)^e + knn'$ for some integer k. Reducing by n' yields $c = \left(v^* r^* u^{-1} r^{-1}\right)^e \bmod n'$. As $\mathcal{B}$ did not query the inversion oracle at all, it thus wins with a probability only polynomially smaller than E_3, contradicting the assumption that $\Pr[E_3]$ is not negligible. The private collision-resistance follows.

Note that the private collision resistance property actually already holds under the standard RSA assumption (not the one-more RSA assumption), as $\mathcal{B}$ never

[13] Formally, in the following $\mathcal{B}$ honestly simulates all calls to CHash', except for a random query where it embeds the challenge; this causes a loss of $1/q_h$ in the success probability, where q_h is the number of oracle queries made by $\mathcal{A}$.

[14] Note that the adversary already has access to the random oracles in its first phase, i.e., before outputting pk^*. In the following, we assume that it never queried the oracle for $\mathcal{H}_{nn'}(\cdot)$ during this phase. Given the super-polynomial number of possible values for n', this only introduces a negligible loss in the following.

queries the inversion oracle. Formally, in the proof, it would abort when receiving a $c \notin \mathbb{Z}^*_{nn'}$, which—by Lemma 2—imposes a polynomial loss.

Uniqueness. We now prove that the above construction is unique using a sequence of games:

Game 0: The original uniqueness game.

Game 1: As Game 0, but the challenger aborts, if the adversary finds randomness $r^* \neq r'^*$, a public key $\mathsf{pk}^* = n$, a message m^*, and a hash h^* such that $\mathsf{CHashCheck}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{CHashCheck}(\mathsf{pk}^*, m^*, r'^*, h^*) = \texttt{true}$. Let this abort event be denoted E_1.

Transition - Game 0 → Game 1: This cannot happen, as RSA (with the given restrictions on e and r) defines a permutation, and random-oracles behave as functions, regardless of the choice of n. $|\Pr[S_0] - \Pr[S_1]| = 0$ follows. See also Lemma 1.

This proves that our construction is unique. □

E.2 A Construction in Gap-Groups

Next, we present our second direct construction in prime order groups equipped with a bilinear map.

Bilinear Maps. Let G_1, G_2 and G_T be three cyclic multiplicative groups with prime order q, generated by g_i, i.e. $G_i = \langle g_i \rangle$ for $i \in \{1, 2, T\}$. Let $e : G_1 \times G_2 \to G_T$ be a bilinear map such that:

1. Bilinearity: $\forall u \in G_1, \forall v \in G_2 : \forall a, b \in \mathbb{Z}^*_q : e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: $\exists u \in G_1, \exists v \in G_2 : e(u, v) \neq 1$, i.e., $e(g_1, g_2) = g_T$.
3. Computability: There is an efficient algorithm that calculates the mapping e.

This construction is derived from the first construction. Essentially, the main idea is to use a DDH oracle to check the correctness of the commitment. Moreover, we require that the DL assumption holds in G_2, i.e., for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that: $\Pr[(G_1, G_2, G_T, e, g_1, g_2, g_T, q) \leftarrow \mathsf{BGGen}(1^\lambda), x \leftarrow \mathbb{Z}_q, x' \leftarrow \mathcal{A}(G_1, G_2, G_T, e, g_1, g_2, g_T, q, g_2^x) : x = x'] \leq \nu(\lambda)$. Clearly, this implies that DL is hard in G_T. Subsequently, we present our constructions where we assume that the **NP**-languages involved in the proofs of knowledge are implicitly defined by the scheme.

Construction 7 (CHET **in Gap-Groups)** *Let $\{\mathcal{H}^k_{\mathbb{Z}^*_q}\}_{k \in \mathcal{K}}$ denote a family of collision-resistant hash functions $\mathcal{H}^k_{\mathbb{Z}^*_q} : \{0,1\}^* \to \mathbb{Z}^*_q$ indexed by a key $k \in \mathcal{K}$ and let* CHET *be defined as:*

CParGen. *The algorithm* CParGen *generates the public parameters in the following way:*

1. Let $(G_1, G_2, G_T, e, g_1, g_2, g_T, q) \leftarrow \mathsf{BGGen}(1^\lambda)$.

2. *Let* $k \leftarrow \mathcal{K}$ *for the hash function.*
3. *Let* $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$.[15]
4. *Return* $((G_1, G_2, G_T, e, g_1, g_2, g_T, q), k, \mathsf{crs})$

CKGen. *The algorithm* CKGen *generates the key pair in the following way:*

1. *Draw random* $x \leftarrow \mathbb{Z}_q^*$. *Set* $h \leftarrow g_2^x$.
2. *Generate* $\pi_{\mathsf{pk}} \leftarrow \text{NIZKPoK}\{(x) : h = g_2^x\}$.
3. *Let* $(\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}}) \leftarrow \mathsf{KGen}_{\mathsf{enc}}(1^\lambda)$.
4. *Return* $((x, \mathsf{sk}_{\mathsf{enc}}), (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}}))$.

CHash. *To hash* m *w.r.t.* $\mathsf{pk}_{\mathsf{ch}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$ *do:*

1. *If* π_{pk} *is not valid, return* $\perp$.
2. *Draw random* $r \leftarrow \mathbb{Z}_q^*$.
3. *Draw random* $\mathsf{etd} \leftarrow \mathbb{Z}_q^*$.
4. *Let* $h' \leftarrow g_2^{\mathsf{etd}}$.
5. *Generate* $\pi_t \leftarrow \text{NIZKPoK}\{(\mathsf{etd}) : h' = g_2^{\mathsf{etd}})\}$.
6. *Encrypt* r*, i.e., let* $C \leftarrow \Pi.\mathsf{Enc}(\mathsf{pk}_{\mathsf{enc}}, r)$.
7. *Let* $p \leftarrow e(g_1^r, h)$.
8. *Generate* $\pi_p \leftarrow \text{NIZKPoK}\{(r) : p = e(g_1^r, h)\}$.
9. *Let* $a \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m)$.
10. *Let* $b \leftarrow p \cdot e(g_1^a, h')$.
11. *Return* $((b, h', \pi_t), (p, C, \pi_p), \mathsf{etd})$.

$\mathsf{CHashCheck}$. *To check whether a given hash* (b, h', π_t) *is valid on input* $\mathsf{pk}_{\mathsf{ch}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$*,* m*,* (p, C, π_p) *do:*

1. *Return* `false`*, if* $p \notin G_T \ \vee \ h' \notin G_2^*$.
2. *If either* π_p*,* π_t*, or* π_{pk} *are not valid, return* $\perp$.
3. *Let* $a \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m, \tau)$.
4. *Return* `true`*, if* $b = p \cdot e(g_1^a, h')$.
5. *Return* `false`.

Adapt. *To find a collision w.r.t.* m*,* m'*, randomness* (p, C, π_p)*, and trapdoor information* etd*, and* $\mathsf{sk}_{\mathsf{ch}} = (x, \mathsf{sk}_{\mathsf{enc}})$ *do:*

1. *If* $\texttt{false} = \mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, m, (p, C, \pi_p), (b, h', \pi_t))$*, return* $\perp$.
2. *Return* $\perp$*, if* $h' \neq g_2^{\mathsf{etd}}$.
3. *Decrypt* C*, i.e.,* $r \leftarrow \Pi.\mathsf{Dec}(\mathsf{sk}_{\mathsf{enc}}, C)$. *If* $r = \perp$*, return* $\perp$.
4. *If* $m = m'$*, return* (p, C, π_p).
5. *Let* $a \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m)$.
6. *Let* $a' \leftarrow \mathcal{H}^k_{\mathbb{Z}_q^*}(m')$.
7. *If* $p \neq e(g_1^r, g_2^x)$*, return* $\perp$.
8. *If* $a = a'$*, return* $r = (p, C, \pi_p)$.
9. *Let* $r' \leftarrow \frac{rx + a \cdot \mathsf{etd} - a' \cdot \mathsf{etd}}{x}$.
10. *Let* $p' \leftarrow e(g_1^{r'}, g_2^x)$.
11. *Encrypt* r'*, i.e., let* $C' \leftarrow \Pi.\mathsf{Enc}(\mathsf{pk}_{\mathsf{enc}}, r')$.
12. *Generate* $\pi'_p \leftarrow \text{NIZKPoK}\{(r') : p' = e(g_1^{r'}, g_2^x)\}$.
13. *Return* (p', C', π'_p).

[15] Actually we need one crs per language, but we do not make this explicit here.

Most of the checks can already be done in advance, e.g., at a PKI, which only generates certificates, if the restrictions on each public key are fulfilled. Moreover, we do not require that the correctness of the ciphertext is proven.

Theorem 6. *If the DL assumption in G_2 holds, $\mathcal{H}^k_{\mathbb{Z}^*_{|G_2|}}$ is collision-resistant, Π is is IND-CCA2 secure, and* NIZKPoK *is secure, then the chameleon-hash with ephemeral trapdoors* CHET *in Construction 7 is secure.*

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

Indistinguishability. Indistinguishability is trivial: as the adversary never sees both the hash and the adapted hash at the same time, the distributions are identical. The proof is therefore omitted.

Public Collision-Resistance. We prove public collision-resistance using a sequence of games.

Game 0: The original public collision-resistance game.

Game 1: As Game 0, but upon setup we obtain $(\mathsf{crs}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$ upon setup, store τ and henceforth simulate all proofs using $\mathcal{S}_2(\mathsf{crs}, \tau, \cdot)$.

Transition - Game 0 → Game 1: A distinguisher between Game 0 and Game 1 is a zero-knowledge distinguisher, i.e., $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{zk}}(\lambda)$.

Game 2: As Game 1, but upon setup we obtain $(\mathsf{crs}, \tau, \xi) \leftarrow \mathcal{S}(1^\lambda)$, and additionally store ξ.

Transition - Game 1 → Game 2: Under simulation sound extractability, this change is conceptual, i.e., $\Pr[S_1] = \Pr[S_2]$.

Game 3: As Game 2, but we simulate the Adapt oracle as follows:

```
Adapt. To find a collision w.r.t. $m$, $m'$, randomness $(p, C, \pi_p)$, and trapdoor
    information etd, and $\mathsf{sk_{ch}} = (x, \mathsf{sk_{enc}})$ do:
    1. If $(p, C, \cdot)$ corresponds to a previous Adapt query, set $\mathsf{AD} = \top$
       and $\mathsf{AD} = \bot$ otherwise. If only $C$ corresponds to a previous
       query, return $\bot$.
    2. If false $= \mathsf{CHashCheck}(\mathsf{pk_{ch}}, m, (p, C, \pi_p), (b, h', \pi_t))$, return $\bot$.
    ...
```

Transition - Game 2 → Game 3: This change is conceptual, i.e., $\Pr[S_2] = \Pr[S_3]$ (observe that C is unconditionally binding, and, thus, modifying p implies that the check $p = e(g_1^x, g_2^r)$ which is performed within Adapt fails, and the oracle would abort anyway).

Game 4: As Game 3, but we further change the Adapt oracle as follows:

```
Adapt. To find a collision w.r.t. $m$, $m'$, randomness $(p, C, \pi_p)$, and trapdoor
    information $\mathsf{etd}$, and $\mathsf{sk_{ch}} = (x, \mathsf{sk_{enc}})$ do:
    1. If $(p, C, \cdot)$ corresponds to a previous Adapt query, set $\mathsf{AD} = \top$
       and $\mathsf{AD} = \bot$ otherwise. If only $C$ corresponds to a previous query,
       return $\bot$.
    ...
    3. $\boxed{\text{If } \mathsf{AD} = \bot,}$ decrypt $C$, i.e., $r \leftarrow \Pi.\mathsf{Dec}(\mathsf{sk_{enc}}, C)$. If $r = \bot$, return
       $\bot$.
    ...
    7. $\boxed{\text{If } \mathsf{AD} = \bot,}$ check if $p \neq e(g_1^r, g_2^x)$, and return $\bot$ if so.
    ...
```

Transition - Game 3 → Game 4: This change is conceptual, i.e., $\Pr[S_3] = \Pr[S_4]$ (the checks are only omitted if we know that they would not yield to an abort).

Game 4: As Game 3, but we further change the Adapt oracle as follows:

```
Adapt. To find a collision w.r.t. $m$, $m'$, $(b, h', \pi_t)$, randomness $(p, C, \pi_p)$,
    and trapdoor information $\mathsf{etd}$, and $\mathsf{sk_{ch}} = (x, \mathsf{sk_{enc}})$ do:
    1. If $(p, C, \cdot)$ corresponds to a previous Adapt query, set $\mathsf{AD} = \top$
       and $\mathsf{AD} = \bot$ otherwise. If only $C$ corresponds to a previous query,
       return $\bot$.
    ...
    3. If $\mathsf{AD} = \bot$, decrypt $C$, i.e., $r \leftarrow \Pi.\mathsf{Dec}(\mathsf{sk_{enc}}, C)$. If $r = \bot$, return
       $\bot$.
    ...
    7. If $\mathsf{AD} = \bot$, check if $p \neq e(g_1^r, g_2^x)$, and return $\bot$ if so.
    8. If $a = a'$, return $(p, C, \pi_p)$.
    9. $\boxed{\mathtt{NOP}}$
    10. Let $\boxed{p' \leftarrow \frac{b}{e(g^{a'}, h')}}$.
    11. $\boxed{C' \leftarrow \Pi.\mathsf{Enc}(\mathsf{pk_{enc}}, \bot)}$ .
    12. Generate $\pi'_p \leftarrow \mathcal{S}_2(\mathsf{crs}, \tau, (p', g_2^x))$.
    ...
```

Transition - Game 3 → Game 4: A distinguisher between Game 3 and Game 4 is an IND-CCA2 distinguisher for Π, i.e., $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{c}}(\lambda)$.

Game 5: As Game 4, but we further modify Adapt so that it runs on an $\mathsf{sk_{ch}}$ where x is replaced by g^x:

```
Adapt. To find a collision w.r.t. $m$, $m'$, $(b, h', \pi_t)$, randomness $(p, C, \pi_p)$,
    and trapdoor information $\mathsf{etd}$, and $\mathsf{sk_{ch}} = (\boxed{g_2^x}, \mathsf{sk_{enc}})$ do:
    ...
```

Transition - Game 4 → Game 5: This change is conceptual, i.e., $\Pr[S_4] = \Pr[S_5]$.

Game 6: As Game 5, but for every query to Adapt, we store (p, C, π_p) if π_p was not previously simulated within Adapt in $\mathtt{R}[(b, h', \pi_t)] \leftarrow (p, C, \pi_p)$. Now, for every forgery either both r^* or r'^* are fresh, or one of them contains a proof π_p (resp. π'_p) which we previously simulated in the Adapt oracle. If one of them contains such a proof, we replace the respective randomness tuple (p, C, π_p) by $\mathtt{R}[h^*]$.

Transition - Game 5 → Game 6: This change is conceptual, i.e., $\Pr[S_5] = \Pr[S_6]$. Observe that the fact that a proof stems from a tuple returned by Adapt implies that a query with a tuple (p, C, π_p) where π_p was not simulated must once have happened. Further, the modified forgery is still a valid public collision freeness forgery.

Game 7: As Game 6, but for the modified forgery we extract both r and r' from π_p and π'_p contained in $r^* = (p, C, \pi_p)$ and $r'^* = (p', C', \pi'_p)$. If the extraction fails, we abort.

Transition - Game 6 → Game 7: Both games proceed identically, unless we have to abort, i.e, $|\Pr[S_6] - \Pr[S_7]| \leq 2 \cdot \nu_{\mathsf{e}}(\lambda)$.[16]

Game 8: As Game 7, but for π_t contained in h^* we extract etd and abort if the extraction fails.

Transition - Game 7 → Game 8: Both games proceed identically, unless we have to abort, i.e., $|\Pr[S_7] - \Pr[S_8]| \leq \nu_{\mathsf{e}}(\lambda)$.

Game 9: As Game 9, but we obtain a DL-challenge $(G_1, G_2, G_T, e, g_1, g_2, g_T, q, g_2^x)$, perform the setup with respect to $G_1, G_2, G_T, e, g_1, g_2, g_T, q)$ and embed g_2^x into $\mathsf{pk}_{\mathsf{ch}}$.

Transition - Game 8 → Game 9: This change is conceptual, i.e., $\Pr[S_8] = \Pr[S_9]$.

In Game 9, for every forgery (modified according to Game 6) we have that $h^* = (b, h', \pi_t)$ contains $b = g_T^{rx + a\mathsf{etd}} = g_T^{r'x + a'\mathsf{etd}}$. Thus, we have $rx + a\mathsf{etd} = r'x + a'\mathsf{etd}$. Hence, x can easily be calculated, which is the solution to the DL-challenge. This extraction is only possible, if $a \neq a'$. However, if this is not the case we have a collision in the hash-function. Taking the union bound yields that $\Pr[S_9] = \nu_{\mathsf{DL}}(\lambda) + \nu_{\mathsf{CR}}(\lambda)$; all intermediate game changes are negligible, which concludes the proof.

Private Collision-Resistance. Below we prove private collision resistance using a sequence of games.

Game 0: The original private collision-resistance game.

Game 1: As Game 0, but upon setup we obtain $(\mathsf{crs}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$ upon setup, store τ and henceforth simulate all proofs using $\mathcal{S}_2(\mathsf{crs}, \tau, \cdot)$.

Transition - Game 0 → Game 1: A distinguisher between Game 0 and Game 1 is a zero-knowledge distinguisher, i.e., $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{zk}}(\lambda)$.

Game 2: As Game 1, but upon setup we obtain $(\mathsf{crs}, \tau, \xi) \leftarrow \mathcal{S}(1^\lambda)$, and additionally store ξ.

Transition - Game 1 → Game 2: Under simulation sound extractability, this change is conceptual, i.e., $\Pr[S_1] = \Pr[S_2]$.

[16] For simplicity we collapsed both extractions in a single game change. It is easy to unroll them into two separate game changes.

Game 3: As Game 2, but we modify the CHash oracle so that it no longer draws etd uniformly at random but directly draws h' uniformly at random from G_2^*.

```
$\mathsf{CHash}$. To hash $m$ w.r.t. $\mathsf{pk_{ch}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk_{enc}})$ do:
    ...
    5. Let $\boxed{h' \leftarrow G_2^*}$.
    ...
```

Game 4: As Game 3, but for every π_p returned by CHash we record the value r so that $p = e(g_1^r, h)$ in $\mathtt{R}[p] \leftarrow r$.

Transition - Game 3 → Game 4: This change is conceptual, i.e., $\Pr[S_3] = \Pr[S_4]$.

Game 5: As Game 4, but for pk^* output by the adversary we extract x so that $g_2^x = h$. If the extraction fails, we abort.

Transition - Game 4 → Game 5: Both games proceed identically, unless we have to abort, i.e, $\Pr[S_4] - \Pr[S_5]| \leq \nu_{\mathsf{e}}(\lambda)$.

Game 6: As Game 5, but we obtain a DL instance $(G_1, G_2, G_T, e, g_1, g_2, g_T, q, g_2^t)$, perform the setup with respect to $(G_1, G_2, G_T, e, g_1, g_2, g_T, q)$ and further modify Chash as follows:

```
$\mathsf{CHash}$. To hash $m$ w.r.t. $\mathsf{pk_{ch}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk_{enc}})$ do:
    ...
    5. Let $\boxed{s \leftarrow \mathbb{Z}_q^*,\ h' \leftarrow (g_2^t)^s}$.
    ...
```

Furthermore, we record $\mathtt{S}[h'] \leftarrow s$.

Transition - Game 5 → Game 6: This change is conceptual, i.e., $\Pr[S_5] = \Pr[S_6]$.

Game 7: As Game 6, but if π_p or π'_p contained in $r^* = (p, C, \pi_p)$ and $r'^* = (p', C', \pi'_p)$ do not correspond to a CHash answer we obtain r and r' using the extractor and set $\mathtt{R}[p] \leftarrow r$ or $\mathtt{R}[p'] \leftarrow r'$. If the extraction fails, we abort.

Transition - Game 6 → Game 7: Both games proceed identical unless we have to abort, i.e., $|\Pr[S_6] - \Pr[S_7]| \leq 2 \cdot \nu_e(\lambda)$.[17]

Now, if the adversary $\mathcal{A}$ outputs $(m^*, r^*, m'^*, r'^*, h^*)$ such that $h^* = p \cdot e(g_1^a, h') = p' \cdot e(g_1^{a'}, h')$ in Game 7, $\mathcal{B}$ proceeds as follows. By definition, we have that $g_T^{rx+ats} = g_T^{r'x+a'ts}$ (Note, $g^{ts} = h'$ in both cases, which, by definition, needs to be returned by the CHash oracle, and thus $s = \mathtt{S}[h']$ is known). It follows that we have $rx + ats = r'x + a'ts$. As all variables but t are now known (the values for r and r' can be obtained from $\mathtt{R}$), we have that t can be calculated and returned as DL solution unless $a = a'$, which would however imply a collision for the hash function. Taking the union bound, yields $\Pr[S_7] \leq \nu_{\mathsf{DL}}(\lambda) + \nu_{\mathsf{CR}}(\lambda)$; all intermediate game changes are negligible, which concludes the proof.

□

Note, we do require CCA2-security, as the adaption algorithm acts as a decryption oracle. Moreover, we do not achieve uniqueness, as the holder of $\mathsf{sk_{ch}}$ can always derive new randomness, e.g., by re-encrypting r_1.

[17] For simplicity we collapsed both extractions in a single game change. It is easy to unroll them into two separate game changes.

F Security Proofs

F.1 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 $b = 0$.

Game 1: As Game 0, but instead of calculating the hash h_1 as in the game, directly hash.

Transition - Game 0 → Game 1: Due to the indistinguishability of the chameleon hashes, this hop only changes the view of the adversary negligibly due to the the indistinguishability of the chameleon hashes. More formally, 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 $\mathsf{pk}_{\mathsf{ch}}^1$, 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 due to the the indistinguishability of the chameleon hashes. More formally, 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 $\mathsf{pk}_{\mathsf{ch}}^2$, 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 $b = 1$. As each hop changes the view only negligibly, indistinguishability is proven. 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 $(m^*, r^*, m'^*, r'^*, h^*)$.

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. A distinguisher $\mathcal{A}$ for this hop can be turned into a forger $\mathcal{B}$ against the collision-resistance of the underlying chameleon-hash. $\mathcal{B}$ proceeds as follows. It receives $\mathsf{pk}_{\mathsf{ch}}^1$ as the challenge public key. It uses this key to initialize $\mathcal{A}$. As the only oracle $\mathcal{B}$ has to simulate is the Adapt-oracle, it proceeds as follows. On input $m^*, m'^*, r^*, \mathsf{etd}^*, h^*$, $\mathcal{B}$ first checks, if the hash verifies. If not, it returns $\perp$. Otherwise, $\mathcal{B}$ computes $r_2' \leftarrow \mathsf{CH.Adapt}(\mathsf{sk}_{\mathsf{ch}}^{2^*}, m^*, m'^*, r^*)$. Adversary $\mathcal{B}$ then queries its own adaption oracle to receive 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", i.e., has never been returned by the Adapt-oracle. Thus, $\mathcal{B}$ can return $(m^*, r_1^*, m'^*, r_1'^*, h_1^*)$ as its own forgery attempt. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{ch\text{-}coll}}(\lambda)$ follows.

As each 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 unique.

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 $(m^*, r^*, m'^*, r'^*, h^*)$.

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. Moreover, let q_h be the number of queries to the hashing oracle. 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, and pk^* from $\mathcal{A}$. Then proceed as follows. Draw $i \leftarrow \{1, 2, \ldots, q_h\}$. For each query $j \neq i$, let $(\mathsf{pk}_{\mathsf{ch}}^{2,i}, \mathsf{sk}_{\mathsf{ch}}^{2,i}) \leftarrow \mathsf{CH.CKGen}(1^\lambda)$. On input m, compute $(h, r) \leftarrow \mathsf{CH.CHash}(\mathsf{pk}_{\mathsf{ch}}^{2,i}, m)$. Otherwise, i.e., $i = j$, on input m, compute $(h, r) \leftarrow \mathsf{CH.CHash}(\mathsf{pk}_{\mathsf{ch}}^2, m)$. Next, let $\mathsf{pk}_{\mathsf{ch}}^{2,j} \leftarrow \mathsf{pk}_{\mathsf{ch}}^2$. In both cases, $\mathcal{B}$ gives $((r', r), (h, \mathsf{pk}_{\mathsf{ch}}^{2,i}))$ to $\mathcal{A}$. At some point, $\mathcal{A}$ returns $(m^*, r^*, m'^*, r'^*, h^*)$. As we know that $h_2^* = h_2'^*$, and m^* must be fresh by assumption, $\mathcal{B}$ can return $(m^*, r_2^*, m'^*, r_2'^*, h_2^*)$ as its own forgery attempt, if the hash returned is the one the challenge was embedded in. Thus, the probability that $\mathcal{B}$ wins is the same as $\mathcal{A}$, divided by q_h, as $\mathcal{B}$ has to guess on which query the adversary $\mathcal{A}$ finds the collision. $|\Pr[S_0] - \Pr[S_1]| \leq q_h \nu_{\mathsf{ch\text{-}coll}}(\lambda)$ follows.

As each 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. Let $\mathcal{A}$ be an adversary which breaks the uniqueness of our construction. We can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to break the uniqueness of the underlying chameleon-hash. In particular, consider the following sequence of games:

Game 0: The original uniqueness game.

Game 1: As Game 0, but we abort if the adversary outputs a randomness $r^* \neq r'^*$, a public key pk^*, a message m, and a hash h^* such that $\mathsf{CHashCheck}(\mathsf{pk}^*, m^*, r^*, h^*) = \mathsf{CHashCheck}(\mathsf{pk}^*, m^*, r'^*, h^*) = \mathtt{true}$.

Transition - Game 0 → Game 1: Let us use E_1 to refer to the abort event. We reduce this case to the uniqueness of the underlying chameleon-hash. $\mathcal{B}$ proceeds as follows. It initializes $\mathcal{A}$ with 1^λ. At some point, $\mathcal{A}$ returns $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*)$. By construction, we know that r^* is of the form (r_1^*, r_2^*), and that h^* of the form $(h_1^*, h_2^*, \mathsf{pk}'^*)$, and r'^* is of the form $(r_1'^*, r_2'^*)$, respectively. Moreover, by assumption, we know that $\mathsf{CH.CHashCheck}(\mathsf{pk}^*, m^*, r_1^*, h_1^*) = \mathsf{CH.CHashCheck}(\mathsf{pk}^*, m^*, r_1'^*, h_1^*) = \mathtt{true}$, but also that $\mathsf{CH.CHashCheck}(\mathsf{pk}'^*, m^*, r_2^*, h_2^*) = \mathsf{CH.CHashCheck}(\mathsf{pk}'^*, m^*, r_2'^*, h_2^*) = \mathtt{true}$. However, we also know that $r_1'^* \neq r_1^*$ or $r_2'^* \neq r_2^*$. Thus, $\mathcal{B}$ can return $(\mathsf{pk}^*, m^*, r_1^*, r_1'^*)$, if $r_1^* \neq r_1'^*$, or $(\mathsf{pk}'^*, m^*, r_2^*, r_2'^*)$, if $r_2'^* \neq r_2^*$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{ch\text{-}unique}}(\lambda)$ follows.

As each game hop only changes the view of the adversary negligibly, this proves that our construction is unique. □

F.2 Two Additional Lemmas

For our proofs, we need two auxiliary lemmas which we prove subsequently. The first one is well-known, while the second one allows an approximation for our reduction to succeed.

Lemma 1. *Let $n \geq 2$ be any arbitrary integer, and $e > n$ be any prime. Then, $r^e \equiv r'^e \bmod n$ implies $r = r'$, if $r \in \mathbb{Z}_n^*$, and $r' \in \mathbb{Z}_n^*$.*

Proof. Because of $e > n \geq \varphi(n)$ and e prime we have that $\gcd(e, \varphi(n)) = 1$. Therefore, there exists d such that $de \equiv 1 \bmod \varphi(n)$. We now have $r^{ed} \equiv r'^{ed} \bmod n$, and the claim follows as $x^{\varphi(n)} \equiv 1 \bmod n$ for all $x \in \mathbb{Z}_n^*$. □

Lemma 2. *There exists a polynomial $p(\cdot)$ such that for every adversary $\mathcal{A}$ that on input n outputs n' of the same bit-length, it holds that:*

$$\Pr\left[y \in \mathbb{Z}_{nn'}^* : (n, p, q, e, d) \leftarrow \mathsf{RSAKGen}(1^\lambda), n' \leftarrow \mathcal{A}(n), y \leftarrow \mathbb{Z}_n^*\right] > \frac{1}{p(\cdot)}, \quad (1)$$

i.e., the given probability is non-negligible.

Before proving the lemma, we recap the following fact:

Fact 1 (Th. 8.8.7 [BS96]) *Let $n \geq 3$ be an integer. It then holds that:*

$$\frac{\varphi(n)}{n} \geq \frac{1}{e^\gamma \log\log n + \frac{3}{\log\log n}},$$

where $\gamma = 0.57721\ldots$ is the Euler-Mascheroni constant.

Now note that this ratio is noticeable in λ for every number n output by an efficient algorithm on input 1^λ, as the bit-length of such an n is polynomially bounded in λ, say by $p(\cdot)$, and thus $n \leq 2^{p(\lambda)}$. Thus, the denominator of the above ratio is bounded by $O(\log(p(\lambda)))$ and thus by $O(\lambda)$.

Proof. First note that $y \in \mathbb{Z}_{nn'}^*$ if and only if $y \in \mathbb{Z}_n^*$ and $y \in \mathbb{Z}_{n'}^*$. The former holds true by definition; we thus can replace $\mathbb{Z}_{nn'}^*$ by $\mathbb{Z}_{n'}^*$ in (1).

If $n = n'$ the sought probability is 1. Otherwise, note that replacing $\mathbb{Z}_n^*$ by $\mathbb{Z}_n$ only changes the probability by a negligible amount, as for an RSA modulus $\varphi(n)/n$ is overwhelming. In the following we now denote the probability space (i.e., $(n, p, q, e, d) \leftarrow \mathsf{RSAKGen}(1^\lambda), n' \leftarrow \mathcal{A}(n), y \leftarrow \mathbb{Z}_n$) by Ω. Furthermore, $\Omega[\![m]\!]$ denotes the same probability space except that $y \leftarrow \mathbb{Z}_n$ is replaced by $y \leftarrow \mathbb{Z}_m$.

Let $n > n'$. We then have:

$$\begin{aligned}
\Pr[y \in \mathbb{Z}_{n'}^* : \Omega] &= \Pr[y \in \mathbb{Z}_{n'}^* \wedge 0 \leq y < n' : \Omega] + \Pr[y \in \mathbb{Z}_{n'}^* \wedge n' \leq y < n : \Omega] \\
&\geq \Pr[y \in \mathbb{Z}_{n'}^* \wedge 0 \leq y < n' : \Omega] \\
&= \Pr[y \in \mathbb{Z}_{n'}^* | 0 \leq y < n' : \Omega] \cdot \Pr[0 \leq y < n' : \Omega] \\
&= \Pr[y \in \mathbb{Z}_{n'}^* : \Omega[\![n']\!]] \cdot \frac{n'}{n} \\
&\geq \frac{\varphi(n')}{n'} \cdot \frac{1}{2} = \frac{1}{O(\lambda)} \,.
\end{aligned}$$

For $n < n'$ we have that:

$$\begin{aligned}
\Pr[y \in \mathbb{Z}_{n'}^* : \Omega] &= \Pr[y \in \mathbb{Z}_{n'}^* | 0 \leq y < n : \Omega[\![n']\!]] \\
&= \Pr[y \in \mathbb{Z}_{n'}^* \wedge 0 \leq y < n : \Omega[\![n']\!]] \cdot \Pr[0 \leq y < n : \Omega[\![n']\!]] \\
&\geq \Pr[y \in \mathbb{Z}_{n'}^* : \Omega[\![n']\!]] \cdot \frac{n}{n'} \\
&\geq \frac{\varphi(n')}{n'} \cdot \frac{1}{2} = \frac{1}{O(\lambda)} \,.
\end{aligned}$$

The claim of the lemma follows. □

F.3 Proof of Theorem 2

For the following proof we assume that the reduction sets up the NIZKPoK-parameters, but also the groups used in the protocol and the hashing-key k. This is not made explicit.

Proof. As before, we only need to prove that our construction is indistinguishable, publicly collision-resistant, and privately collision-resistant. Again, we prove each property on its own.

Indistinguishability. Indistinguishability is trivial: as adversary never sees both the hash and the adapted hash at the same time the distributions are equivalent.

Public Collision-Resistance. We prove public collision-resistance using a sequence of games.

Game 0: The original public collision-resistance game.

Game 1: As Game 0, but upon setup we obtain $(\mathsf{crs}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$ upon setup, store τ and henceforth simulate all proofs using $\mathcal{S}_2(\mathsf{crs}, \tau, \cdot)$.

Transition - Game 0 → Game 1: A distinguisher between Game 0 and Game 1 is a zero-knowledge distinguisher, i.e., $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{zk}}(\lambda)$.

Game 2: As Game 1, but upon setup we obtain $(\mathsf{crs}, \tau, \xi) \leftarrow \mathcal{S}(1^\lambda)$, and additionally store ξ.

Transition - Game 1 → Game 2: Under simulation sound extractability, this change is conceptual, i.e., $\Pr[S_1] = \Pr[S_2]$.

Game 3: As Game 2, but we simulate the Adapt oracle as follows:

```
Adapt. To find a collision w.r.t. $m$, $m'$, $(b, h', \pi_t)$, randomness $(p, C, \pi_p)$,
    and trapdoor information $\mathsf{etd}$, and $\mathsf{sk_{ch}} = (x, \mathsf{sk_{enc}})$ do:
    1. $\boxed{\text{If } (p, C, \cdot) \text{ corresponds to a previous } \mathsf{Adapt} \text{ query, set } \mathsf{AD} = \top}$
       $\boxed{\text{and } \mathsf{AD} = \bot \text{ otherwise. If only } C \text{ corresponds to a previous}}$
       $\boxed{\text{query, return } \bot.}$
    2. If $\mathtt{false} = \mathsf{CHashCheck}(\mathsf{pk_{ch}}, m, (p, C, \pi_p), (b, h', \pi_t))$, return $\bot$.
    ...
```

Transition - Game 2 → Game 3: This change is conceptual, i.e., $\Pr[S_2] = \Pr[S_3]$ (observe that C is unconditionally binding, and, thus, modifying p implies that the check $p = g^{xr}$ which is performed within Adapt fails, and the oracle would abort anyway).

Game 4: As Game 3, but we further change the Adapt oracle as follows:

```
Adapt. To find a collision w.r.t. $m$, $m'$, $(b, h', \pi_t)$, randomness $(p, C, \pi_p)$,
    and trapdoor information $\mathsf{etd}$, and $\mathsf{sk_{ch}} = (x, \mathsf{sk_{enc}})$ do:
    1. If $(p, C, \cdot)$ corresponds to a previous Adapt query, set $\mathsf{AD} = \top$
       and $\mathsf{AD} = \bot$ otherwise. If only $C$ corresponds to a previous query,
       return $\bot$.
    ...
    3. $\boxed{\text{If } \mathsf{AD} = \bot,}$ decrypt $C$, i.e., $r \leftarrow \mathit{\Pi}.\mathsf{Dec}(\mathsf{sk_{enc}}, C)$. If $r = \bot$, return
       $\bot$.
    ...
    7. $\boxed{\text{If } \mathsf{AD} = \bot,}$ check if $p \neq g^{xr}$, and return $\bot$ if so.
    ...
```

Transition - Game 3 → Game 4: This change is conceptual, i.e., $\Pr[S_3] = \Pr[S_4]$ (the checks are only omitted if we know that they would not yield to an abort).

Game 4: As Game 3, but we further change the Adapt oracle as follows:

```
Adapt. To find a collision w.r.t. $m$, $m'$, $(b, h', \pi_t)$, randomness $(p, C, \pi_p)$,
  and trapdoor information $\mathsf{etd}$, and $\mathsf{sk_{ch}} = (x, \mathsf{sk_{enc}})$ do:
  1. If $(p, C, \cdot)$ corresponds to a previous Adapt query, set $\mathsf{AD} = \top$
     and $\mathsf{AD} = \bot$ otherwise. If only $C$ corresponds to a previous query,
     return $\bot$.
  ...
  3. If $\mathsf{AD} = \bot$, decrypt $C$, i.e., $r \leftarrow \Pi.\mathsf{Dec}(\mathsf{sk_{enc}}, C)$. If $r = \bot$, return
     $\bot$.
  ...
  7. If $\mathsf{AD} = \bot$, check if $p \neq g^{xr}$, and return $\bot$ if so.
  8. If $a = a'$, return $(p, C, \pi_p)$.
  9. $\boxed{\mathtt{NOP}}$
  10. Let $\boxed{p' \leftarrow \frac{b}{g^{a'\mathsf{etd}}}}$.
  11. $\boxed{C' \leftarrow \Pi.\mathsf{Enc}(\mathsf{pk_{enc}}, \bot)}$ .
  12. Generate $\pi'_p \leftarrow \mathcal{S}_2(\mathsf{crs}, \tau, (p', h))$.
  ...
```

Transition - Game 3 → Game 4: A distinguisher between Game 3 and Game 4 is an IND-CCA2 distinguisher for Π, i.e., $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{c}}(\lambda)$.

Game 5: As Game 4, but we further modify Adapt so that it runs on an $\mathsf{sk_{ch}}$ where x is replaced by g^x:

```
Adapt. To find a collision w.r.t. $m$, $m'$, $(b, h', \pi_t)$, randomness $(p, C, \pi_p)$,
  and trapdoor information $\mathsf{etd}$, and $\mathsf{sk_{ch}} = (\boxed{g^x}, \mathsf{sk_{enc}})$ do:
  ...
  7. If $\mathsf{AD} = \bot$, check if $p \neq (\boxed{g^x})^r$, and return $\bot$ if so.
  ...
```

Transition - Game 4 → Game 5: This change is conceptual, i.e., $\Pr[S_4] = \Pr[S_5]$.

Game 6: As Game 5, but for every query to Adapt, we store (p, C, π_p) if π_p was not previously simulated within Adapt in $\mathtt{R}[(b, h', \pi_t)] \leftarrow (p, C, \pi_p)$. Now, for every forgery either both r^* or r'^* are fresh, or one of them contains a proof π_p (resp. π'_p) which we previously simulated in the Adapt oracle. If one of them contains such a proof, we replace the respective randomness tuple (p, C, π_p) by $\mathtt{R}[h^*]$.

Transition - Game 5 → Game 6: This change is conceptual, i.e., $\Pr[S_5] = \Pr[S_6]$. Observe that the fact that a proof stems from a tuple returned by Adapt implies that a query with a tuple (p, C, π_p) where π_p was not simulated must once have happened. Further, the modified forgery is still a valid public collision freeness forgery.

Game 7: As Game 6, but for the modified forgery we extract both r and r' from π_p and π'_p contained in $r^* = (p, C, \pi_p)$ and $r'^* = (p', C', \pi'_p)$. If the extraction fails, we abort.

Transition - Game 6 → Game 7: Both games proceed identically, unless we have to abort, i.e, $\Pr[S_6] - \Pr[S_7]| \leq 2 \cdot \nu_{\mathsf{e}}(\lambda)$.[18]

Game 8: As Game 7, but for π_t contained in h^* we extract etd and abort if the extraction fails.

Transition - Game 7 → Game 8: Both games proceed identically, unless we have to abort, i.e., $|\Pr[S_7] - \Pr[S_8]| \leq \nu_{\mathsf{e}}(\lambda)$.

Game 9: As Game 8, but we obtain a DL-challenge (G, g, q, g^x), perform the setup with respect to (G, g, q) and embed g^x into $\mathsf{pk}_{\mathsf{ch}}$.

Transition - Game 8 → Game 9: This change is conceptual, i.e., $\Pr[S_8] = \Pr[S_9]$.

In Game 9, for every forgery (modified according to Game 6) we have that $h^* = (b, h', \pi_t)$ contains $b = g^{rx+a\mathsf{etd}} = g^{r'x+a'\mathsf{etd}}$. Thus, we have $rx + a\mathsf{etd} = r'x + a'\mathsf{etd}$. Hence, x can easily be calculated, which is the solution to the DL-challenge. This extraction is only possible, if $a \neq a'$. However, if this is not the case we have a collision in the hash-function. Taking the union bound yields that $\Pr[S_9] = \nu_{\mathsf{DL}}(\lambda) + \nu_{\mathsf{CR}}(\lambda)$; all intermediate game changes are negligible, which concludes the proof.

Private Collision-Resistance. Below we prove private collision resistance using a sequence of games.

Game 0: The original private collision-resistance game.

Game 1: As Game 0, but upon setup we obtain $(\mathsf{crs}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$ upon setup, store τ and henceforth simulate all proofs using $\mathcal{S}_2(\mathsf{crs}, \tau, \cdot)$.

Transition - Game 0 → Game 1: A distinguisher between Game 0 and Game 1 is a zero-knowledge distinguisher, i.e., $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{zk}}(\lambda)$.

Game 2: As Game 1, but upon setup we obtain $(\mathsf{crs}, \tau, \xi) \leftarrow \mathcal{S}(1^\lambda)$, and additionally store ξ.

Transition - Game 1 → Game 2: Under simulation sound extractability, this change is conceptual, i.e., $\Pr[S_1] = \Pr[S_2]$.

Game 3: As Game 2, but we modify the CHash oracle so that it no longer draws etd uniformly at random but directly draws h' uniformly at random from G^*.

```
CHash. To hash $m$ w.r.t. $\mathsf{pk}_{\mathsf{ch}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$ do:
    ...
    5. Let $\boxed{h' \leftarrow G^*}$.
    ...
```

Game 4: As Game 3, but for every π_p returned by CHash we record the value r so that $p = h^r$ in $\mathtt{R}[p] \leftarrow r$.

Transition - Game 3 → Game 4: This change is conceptual, i.e., $\Pr[S_3] = \Pr[S_4]$.

Game 5: As Game 4, but for pk^* output by the adversary we extract x so that $g^x = h$. If the extraction fails, we abort.

Transition - Game 4 → Game 5: Both games proceed identically, unless we have to abort, i.e, $\Pr[S_4] - \Pr[S_5]| \leq \nu_{\mathsf{e}}(\lambda)$.

[18] For simplicity we collapsed both extractions in a single game change. It is easy to unroll them into two separate game changes.

Game 6: As Game 5, but we obtain a DL instance (G, g, q, g^t), perform the setup with respect to (G, g, q) and further modify CHash as follows:

```
CHash. To hash $m$ w.r.t. $\mathsf{pk_{ch}} = (h, \pi_{\mathsf{pk}}, \mathsf{pk_{enc}})$ do:
    ...
    5. Let $\boxed{s \leftarrow \mathbb{Z}_q^*,\ h' \leftarrow (g^t)^s}$.
    ...
```

Furthermore, we record $\mathtt{S}[h'] \leftarrow s$.

Transition - Game 5 → Game 6: This change is conceptual, i.e., $\Pr[S_5] = \Pr[S_6]$.

Game 7: As Game 6, but if π_p or π'_p contained in $r^* = (p, C, \pi_p)$ and $r'^* = (p', C', \pi'_p)$ do not correspond to a CHash answer we obtain r and r' using the extractor and set $\mathtt{R}[p] \leftarrow r$ or $\mathtt{R}[p'] \leftarrow r'$. If the extraction fails, we abort.

Transition - Game 6 → Game 7: Both games proceed identical unless we have to abort, i.e., $|\Pr[S_6] - \Pr[S_7]| \leq 2 \cdot \nu_e(\lambda)$.[19]

Now, if the adversary $\mathcal{A}$ outputs $(m^*, r^*, m'^*, r'^*, h^*)$ such that $h^* = g^{xr}h'^{a} = g^{xr'}h'^{a'}$ in Game 7, $\mathcal{B}$ proceeds as follows. By definition, we have that $g^{rx+ats} = g^{r'x+a'ts}$ (Note, $g^{ts} = h'$ in both cases, which, by definition, needs to be returned by the CHash oracle, and thus $s = \mathtt{S}[h']$ is known). It follows that we have $rx + ats = r'x + a'ts$. As all variables but t are now known (the values for r and r' can be obtained from $\mathtt{R}$), we have that t can be calculated and returned as DL solution unless $a = a'$, which would however imply a collision for the hash function. Taking the union bound, yields $\Pr[S_7] \leq \nu_{\mathsf{DL}}(\lambda) + \nu_{\mathsf{CR}}(\lambda)$; all intermediate game changes are negligible, which concludes the proof.

□

[19] For simplicity we collapsed both extractions in a single game change. It is easy to unroll them into two separate game changes.