Policy-Based Sanitizable Signatures[‡]

Kai Samelin[1] and Daniel Slamanig[2]

[1] TÜV Rheinland i-sec GmbH, Hallbergmoos, Germany
kaispapers@gmail.com
[2] AIT Austrian Institute of Technology, Vienna, Austria
daniel.slamanig@ait.ac.at

Abstract. Sanitizable signatures allow a single, and signer-defined, sanitizer to modify signed messages in a controlled way without invalidating the respective signature. They turned out to be a fascinating primitive, proven by different variants and extensions, e.g., allowing multiple sanitizers or adding new sanitizers one-by-one. Still, existing constructions are very limited regarding their flexibility in specifying potential sanitizers. In this paper, we propose a different and more powerful approach: Instead of using the sanitizers' public keys directly, we assign attributes to them. Sanitizing is then based on policies, i.e., access structures defined over attributes. A sanitizer can sanitize, if, and only if, it holds a secret key to attributes satisfying the policy associated to a signature, while offering full-scale accountability.

1 Introduction

Unforgeability of a digital signature scheme prevents deriving signatures for a message not explicitly endorsed by the signer. This is a desired property in many use cases and applications of signatures. However, it turned out that certain *controlled* modifications of a signed message can be beneficial in many scenarios [2, 11, 28, 37]. Over the years, different types of signature schemes supporting such modifications have been proposed, including homomorphic signatures [2, 12], redactable signatures [29, 40, 49], and sanitizable signatures [3, 13, 15]. In this paper, we focus on sanitizable signatures (3S henceforth). In a nutshell, a *standard* 3S [3] allows for altering signer-chosen (so called admissible) blocks of signed messages by a *single* semi-trusted entity, called the sanitizer, which is specified by the signer when generating the signature. The sanitizer holds its own public and secret key pair. By using the secret key, the sanitizer can derive modified messages with admissible blocks arbitrarily updated, along with the corresponding valid signatures. Moreover, given a sanitizable signature, there is a (virtual) entity, dubbed the judge, who can determine whether a signature comes from the original signer or has been sanitized, providing accountability. Even though allowing arbitrary modification of signer-specified blocks seems to give too much power to the sanitizer, 3Ss have proven to be useful in numerous use-cases, as exhaustively discussed by Bilzhause et al. [11].

After 3Ss were introduced by Ateniese et al. [3], they received a lot of attention in the recent past. The first thorough security model was given by Brzuska et al. [13] (later slightly modified by Gong et al. [38]). Their work was later extended for multiple signers/sanitizers [14, 23], unlinkability (meaning a derived signatures cannot be linked to its origin) [15, 17–19, 35], non-interactive public-accountability (every party can determine which party is accountable for a given valid message/signature pair) [16], limiting the sanitizer to signer-chosen values [22, 31], invisibility (meaning that an outsider cannot determine which blocks of a message are sanitizable) [6, 19, 20, 34], the case of strongly unforgeable signatures [45], and generalizations such as merging the functionality from sanitizable and redactable signatures [43, 44]. All these extensions make 3Ss suitable for an even broader field of use-cases of (cf. [11] for a discussion), and are directly applicable to our contribution.

In all of the aforementioned work on sanitizable signatures, the sanitizer(s) need(s) to be known *in advance* at signature generation, and there is no possibility to control sanitizing capabilities in a fine-grained way. We note that there is the concept of trapdoor 3Ss [24, 46, 51]. Although here the signer can grant the possibility to sanitize to different entities even after generating the initial signature, existing constructions do either not provide accountability, a central feature of 3S, or require to obtain the trapdoor from the original signer before sanitizing [46]. This drastically restricts the applicability of 3Ss, their flexibility, and may lead to severe problems when the specified sanitizer is not available.

[‡] The project leading to this work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 783119 SECREDAS.

Motivation and Applications. To illustrate the problem, let us consider an enterprise scenario where policies are associated to different types of documents and documents of some type can be sanitized if the person performing the sanitization fullfills the respective policy. For simplicity, assume that sanitizing should be possible if the sanitizer satisfies the policy $P = (\texttt{IT department} \wedge \texttt{admin}) \vee (\texttt{group leader})$. Now, lets say that the head of IT department (the "group-manager") has previously signed a document, e.g., an order, which urgently needs to be sent to reseller but some information needs to be sanitized before, e.g., fixing the number of new SSDs ordered. Unfortunately, the original signer is not available, e.g., due to vacation. Now, everyone satisfying P should be able to sanitize. Since this covers a potentially large set of persons, there is no availability issue, and the document can be sent in time. Still, the department head can control via P who is trusted to sanitize the document if required, and there must be means to determine who did the sanitization in case of a dispute. Realizing this scenario with the state-of-the-art 3S, such as using a sanitizer key per policy and giving the key to everyone satisfying it clearly destroys accountability, i.e., there is no means identifying the accountable party later on, and thus no satisfying solution can be achieved. To tackle this situation, we introduce a primitive denoted policy-based sanitizable signatures (P3S), that allows to sanitize if, and only if, the attributes associated to a sanitizer satisfy the policy associated to the signature, while at the same time providing accountability. We also want to discuss one application of P3S extending the scope of the one discussed in [30]. In particular, [30] discusses an application to updating/rewriting transactions (or more general objects) in blockchains by selectively replacing the hash function used to aggregate transactions (e.g., within a Merkle-tree) by a novel chameleon hash. Now, everyone who wants a transaction that can be updated/rewritten can distribute attribute-keys to users who can potentially update the transactions of this entity. Using P3S instead of this novel chameleon hash to not only hash transactions/objects but combine it with a signature (as usual for transactions and typically also for other objects in blockchains), we achieve stronger guarantees than in [30]. In addition to transparency, meaning that no outsider sees whether updates happened (as also achieved in [30]), using P3S provides accountability, i.e., it can be determined who conducted the update.

Contribution and Our Techniques. In this paper we introduce the notion of policy-based sanitizable signatures (P3S). This primitive provides the functionality that sanitizable signatures are produced with respect to sanitizing policies, sanitizers can obtain secret keys with respect to attributes and can sanitize if, and only if, the signature's policy is satisfied with the attributes in the sanitizer's key. As mentioned above, although a potentially huge set of sanitizers can thus produce a sanitized version, the primitive still provides accountability, i.e., the responsible sanitizer can always be found — if required. We first provide a natural formal framework for P3S by extending the one for 3S. Here it must be noted that in the case of P3S with a potentially large sets of sanitizers and different sanitization keys (depending on attributes) make the formal definition much trickier and somewhat involved. Still, we believe that our proposed definitions are clean and easy to comprehend. Our P3S framework also allows for different *groups* and users can obtain new secret keys in a dynamic fashion. The idea is that signers and sanitizers should be able to re-use their keys across different groups, e.g, in an enterprise every employee may hold a key-pair and could participate in different "sanitizing groups" for different types of documents without re-generating fresh keys for every such group. This is modeled in the vein of dynamic group signatures [9], where we also consider a notion analogous to opening-soundness [48]. Moreover, we propose very strict privacy notions, where even (most of) the keys are generated by the adversary, further strengthening already existing definitions [27, 33, 45].

Then, we provide a construction of P3S which we rigorously analyze in the proposed framework. Technically, the heart of our our construction is a recent primitive called policy-based chameleon hash (PCH) [30], which is a trapdoor collision-resistant hash-function, where the hash computation in addition to the message takes a description of a policy as input. Loosely speaking there can be many different trapdoors and collisions can be found if, and only if, a trapdoor satisfying the policy used for the computation of the hash is known. Looking ahead, the PCH proposed in [30] combines chameleon-hashes with ephemeral trapdoors (CHET) [20] and CCA2 secure ciphertext-policy attributes encryption (CP-ABE). In contrast to the original PCH definition in [30], however, we have to make some minor, yet important, alterations and show that a modified construction from [30] satisfies our stronger notions. In this regard, we also strengthen the CH and CHET definitions by Camenisch et al. [20] to also cover keys generated by the adversary. We believe that this strengthened definitions are also useful in many other scenarios. The concrete PCH construction then requires some additional tools and tricks; In order to achieve accountability, we use an $\vee$-"trick", and attach a non-interactive zero-knowledge proof of knowledge, demonstrating that either the signer or a sanitizer performed the signing, or the sanitization, respectively. The expressiveness of the policies supported by the $\mathsf{P3S}$ are determined by that of the PCH and in particular by that of the underlying CP-ABE scheme. We chose to build upon the existing PCH framework which covers (monotone) access structures as policies as this seems to be the most interesting setting for practical applications.[1] For a detailed intuition on the construction, see Section 4.

2 Preliminaries

Notation. With $\kappa \in \mathbb{N}$ we denote our security parameter. All algorithms implicitly take 1^κ as an additional input. We write $a \leftarrow A(x)$ if a is assigned to the output of algorithm A with input x. An algorithm is efficient, if it runs in probabilistic polynomial time (PPT) in the length of its input. All algorithms are PPT, if not explicitly mentioned otherwise. If we make the random coins r explicit, we use the notation $a \leftarrow A(x;r)$. Otherwise, we assume that the random coins are drawn internally. For $m = (m^1, m^2, \ldots, m^\ell)$, we call m^i a block. Most algorithms may return a special error symbol $\perp \notin \{0,1\}^*$, denoting an exception. Returning output ends execution of an algorithm or an oracle. If S is a set, we write $a \leftarrow_r S$ to denote that a is chosen uniformly at random from S. For a list we require that there is an injective, and efficiently reversible, encoding, mapping the list to $\{0,1\}^*$. 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 $\nu(n) \leq n^{-k}$, $\forall n > n_0$.

Assumptions and Primitives. For our construction to work, we need one-way functions (denoted by f), an unforgeable digital signature scheme $\Sigma = \{\mathsf{PPGen}_\Sigma, \mathsf{KGen}_\Sigma, \mathsf{Sign}_\Sigma, \mathsf{Verf}_\Sigma\}$, an IND-CPA and key-verifiable secure encryption scheme $\Pi = \{\mathsf{PPGen}_\Pi, \mathsf{KGen}_\Pi, \mathsf{Enc}_\Pi, \mathsf{Dec}_\Pi, \mathsf{KVrf}_\Pi\}$. Key-verifiability means that for each public key exactly one secret key exists (e.g., ElGamal suffices), while KVrf_Π checks whether a given secret key sk belongs to a pk. Moreover, we require a non-interactive zero-knowledge proof of knowledge system $\Omega = \{\mathsf{PPGen}_\Omega, \mathsf{Prove}_\Omega, \mathsf{Verify}_\Omega\}$, and a recent primitive dubbed policy-based chameleon-hash (PCH), recently introduced by Derler et al. [30].

Definition 1 (One-Way Functions). *A function $f : D_f \to R_f$ is κ-one-way, if for every PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\Pr[x \leftarrow_r D_f, x' \leftarrow_r \mathcal{A}(f(x)) : f(x) = f(x')] \leq \nu(\kappa)$$

We assume that D_f and R_f are implicitly defined by f.

Definition 2 (Digital Signatures). *A digital signature scheme Σ consists of four algorithms $\{\mathsf{PPGen}_\Sigma$, $\mathsf{KGen}_\Sigma, \mathsf{Sign}_\Sigma, \mathsf{Verf}_\Sigma\}$ such that:*

PPGen_Σ**.** *The algorithm PPGen_Σ outputs the public parameters*

$$\mathsf{pp}_\Sigma \leftarrow_r \mathsf{PPGen}_\Sigma(1^\kappa)$$

We assume that pp_Σ contains 1^κ and is implicit input to all other algorithms.

KGen_Σ**.** *The algorithm KGen_Σ outputs the public and private key of the signer, where κ is the security parameter:*

$$(\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma) \leftarrow_r \mathsf{KGen}_\Sigma(\mathsf{pp}_\Sigma)$$

[1] PCHs and $\mathsf{P3S}$ could be defined for richer policies, e.g., polynomial sized circuits.

Sign_Σ**.** *The algorithm* Sign_Σ *gets as input the secret key* sk_Σ *and the message* $m \in \mathcal{M}$ *to sign. It outputs a signature:*

$$\sigma \leftarrow_r \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, m)$$

Verf_Σ**.** *The deterministic algorithm* Verf_Σ *outputs a decision bit* $d \in \{0, 1\}$*, indicating if the signature* σ *is valid, w.r.t.* pk_Σ *and* m*:*

$$d \leftarrow \mathsf{Verf}_\Sigma(\mathsf{pk}_\Sigma, m, \sigma)$$

For each Σ it is required that the correctness properties hold. In particular, it is required that for all $\kappa \in \mathbb{N}$, for all $\mathsf{pp}_\Sigma \leftarrow_r \mathsf{PPGen}_\Sigma(1^\kappa)$, for all $(\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma) \leftarrow_r \mathsf{KGen}_\Sigma(\mathsf{pp}_\Sigma)$, for all $m \in \mathcal{M}$, $\mathsf{Verf}_\Sigma(\mathsf{pk}_\Sigma, m, \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, m)) = 1$ is true. This definition captures perfect correctness.

We require existential unforgeability (eUNF-CMA) of digital signature schemes. In a nutshell, unforgeability requires that an adversary $\mathcal{A}$ cannot (except with negligible probability) come up with a signature for a message m^* for which the adversary did not see any signature before. As usual, the adversary $\mathcal{A}$ can adaptively query for signatures on messages of its own choice.

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

Fig. 1: Σ Unforgeability

Definition 3 (Σ Unforgeability). *We say a* Σ *scheme is unforgeable, if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* ν *such that:*

$$\Pr\left[\mathbf{Exp}_{\mathcal{A},\Sigma}^{\mathsf{eUNF\text{-}CMA}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 1.

For our definition of public key encryption we need an additional algorithm KVrf_Π verifying if a given key pair is valid along with a corresponding security notion requiring that even for adversarially chosen public keys one can find at most one corresponding secret key.

Definition 4 (Public-Key Encryption). *A public-key encryption-scheme* Π *consists of five algorithms* $\{\mathsf{PPGen}_\Pi, \mathsf{KGen}_\Pi, \mathsf{Enc}_\Pi, \mathsf{Dec}_\Pi, \mathsf{KVrf}_\Pi\}$

PPGen_Π**.** *The algorithm* PPGen_Π *outputs the public parameters of the scheme:*

$$\mathsf{pp}_\Pi \leftarrow_r \mathsf{PPGen}_\Pi(1^\kappa)$$

It is assumed that pp_Π *is implicit input to all other algorithms. Also, this algorithm may be omitted, if it is clear from the context.*

KGen_Π. *The algorithm* KGen_Π *outputs the public and private key, on input* pp_Π*:*

$$(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) \leftarrow_r \mathsf{KGen}_\Pi(\mathsf{pp}_\Pi)$$

Enc_Π. *The algorithm* Enc_Π *gets as input the public key* pk_Π*, and a message* $m \in \mathcal{M}$ *to encrypt. It outputs a ciphertext:*

$$c \leftarrow_r \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, m)$$

Dec_Π. *The deterministic algorithm* Dec_Π *outputs a message* m *(or* $\perp$*, if the ciphertext is invalid) on input* sk_Π*, and a ciphertext* c*:*

$$m \leftarrow \mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, c)$$

KVrf_Π. *The deterministic algorithm* KVrf_Π *decides whether a given secret key* sk_Π *belongs to* pk_Π*, outputting a decision bit* $b \in \{1, 0\}$*.*

$$b \leftarrow \mathsf{KVrf}_\Pi(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi)$$

For each Π, the usual correctness properties must hold. In particular, it is required that for all $\kappa \in \mathbb{N}$, for all $\mathsf{pp}_\Pi \leftarrow_r \mathsf{PPGen}_\Pi(1^\kappa)$, for all $(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) \leftarrow_r \mathsf{KGen}_\Pi(\mathsf{pp}_\Pi)$, for all $m \in \mathcal{M}$, it holds that $\mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, m)) = m$ and $\mathsf{KVrf}_\Pi(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) = 1$ are true.

Moreover, we require that the encryption scheme is Π is IND-CPA-secure and key-verifiable.

```
$\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\Pi}(\kappa)$
  $\mathsf{pp}_\Pi \leftarrow_r \mathsf{PPGen}_\Pi(1^\kappa)$
  $(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) \leftarrow_r \mathsf{KGen}_\Pi(\mathsf{pp}_\Pi)$
  $b \leftarrow_r \{0, 1\}$
  $((m_0^*, m_1^*), state_\mathcal{A}) \leftarrow_r \mathcal{A}(\mathsf{pk}_\Pi)$
  If $|m_0^*| \neq |m_1^*| \vee m_0^* \notin \mathcal{M} \vee m_1^* \notin \mathcal{M}$:
    $c^* \leftarrow \perp$
  Else:
    $c^* \leftarrow_r \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, m_b^*)$
  $b^* \leftarrow_r \mathcal{A}(state_\mathcal{A}, c^*)$
  return 1, if $b^* = b$
  return 0
```

Fig. 2: Π $\mathsf{IND\text{-}CPA}$ Security

Definition 5 **(Π IND-CPA-Security).** *An encryption scheme* Π *is* IND-CPA-*secure, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\left|\Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\Pi}(\kappa) = 1\right] - \tfrac{1}{2}\right| \leq \nu(\kappa)$$

The corresponding experiment is depicted in Figure 2.

Definition 6 **(Π Key-Verifiability).** *An encryption scheme* Π *is key-verifiable, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{Key\text{-}Verifiability}}_{\mathcal{A},\Pi}(\kappa) = 1\right] \leq \nu(\kappa)$$

The corresponding experiment is depicted in Figure 3.

```
$\mathbf{Exp}^{\mathsf{Key\text{-}Verifiability}}_{\mathcal{A},\Pi}(\kappa)$
  $\mathsf{pp}_\Pi \leftarrow_r \mathsf{PPGen}_\Pi(1^\kappa)$
  $(\mathsf{sk}^*_0, \mathsf{sk}^*_1, \mathsf{pk}^*) \leftarrow_r \mathcal{A}(\mathsf{pp}_\Pi)$
  return 0, if $\mathsf{KVrf}_\Pi(\mathsf{sk}^*_0, \mathsf{pk}^*) = 0 \ \vee\ \mathsf{KVrf}_\Pi(\mathsf{sk}^*_1, \mathsf{pk}^*) = 0$
  return 1, if $\mathsf{sk}^*_0 \neq \mathsf{sk}^*_1$
  return 0
```

Fig. 3: Π Key-Verifiability

L be an NP-language with associated witness relation R, i.e., such that $L = \{x \mid \exists w : R(x, w) = 1\}$. In a nutshell, a non-interactive zero-knowledge proof of knowledge allows to verify that the generator of such a proof knows a witness w for some statement x without revealing that witness. More formally, such a system is defined as follows.

Definition 7 (Non-Interactive Zero-Knowledge Proof of Knowledge System). *A non-interactive zero-knowledge proof of knowledge system Ω consists of three algorithms $\{\mathsf{PPGen}_\Omega, \mathsf{Prove}_\Omega, \mathsf{Verify}_\Omega\}$, such that:*

PPGen_Ω. *The algorithm crs_Ω outputs public parameters of the scheme, where κ is the security parameter:*

$$\mathsf{crs}_\Omega \leftarrow_r \mathsf{PPGen}_\Omega(1^\kappa, L)$$

For simplicity, it assumed that crs_Ω is an implicit input to all other algorithms, while the language L is clear from the context.

Prove_Ω. *The algorithm Prove_Ω outputs the proof π, on input of the statement x to be proven, and the corresponding witness w:*

$$\pi \leftarrow_r \mathsf{Prove}_\Omega(x, w)$$

Verify_Ω. *The deterministic algorithm Verify_Ω verifies the proof π, w.r.t. to some statement x, where $d \in \{0, 1\}$:*

$$d \leftarrow \mathsf{Verify}_\Omega(x, \pi)$$

In the context of (zero-knowledge) proof-systems, correctness is sometimes also referred to as completeness. More precisely, it is required that for all $\kappa \in \mathbb{N}$, for all "suitable" L, for all $\mathsf{crs}_\Omega \leftarrow_r \mathsf{PPGen}_\Omega(1^\kappa, L)$, for all $x \in L$, for all w such that $R(x, w) = 1$, for all $\pi \leftarrow_r \mathsf{Prove}_\Omega(x, w)$, it must hold that $\mathsf{Verify}_\Omega(\mathsf{crs}_\Omega, x, \pi) = 1$. Two different security notions are required, i.e., zero-knowledge and simulation-sound extractability, taken from Groth [39].

In a nutshell, zero-knowledge says that the receiver of the proof π does not learn anything except the validity of the statement. It is assumed that the distribution of crs_Ω output by SIM_1 is distributed identically to PPGen_Ω.

Definition 8 (Ω Zero-Knowledge). *A non-interactive proof system Ω is zero-knowledge, if for a fixed language L, for any PPT adversary $\mathcal{A}$, there exists an PPT simulator $\mathsf{SIM} = (\mathsf{SIM}_1, \mathsf{SIM}_2)$ such that there exists a negligible function ν such that:*

$$\left|\Pr\left[\mathbf{Exp}^{\mathsf{Zero\text{-}Knowledge}}_{\mathcal{A},\Omega,\mathsf{SIM},L}(\kappa) = 1\right] - \tfrac{1}{2}\right| \leq \nu(\kappa)$$

The corresponding experiment is depicted in Figure 4. Here, τ is the trapdoor for the simulation.

Simulation-sound extractability says that an adversary cannot generate a proof π^* for a statement it does not know a witness for, while the proof-system is also of knowledge, i.e., the witness w can be extracted from any non-simulated proof π. Clearly, this also implies that the proof-system is non-malleable.

```
$\mathbf{Exp}^{\mathsf{Zero\text{-}Knowledge}}_{\mathcal{A},\Omega,\mathsf{SIM},L}(\kappa)$
  $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa, L)$
  $b \leftarrow_r \{0,1\}$
  $b^* \leftarrow_r \mathcal{A}^{P_b(\cdot,\cdot)}(\mathsf{crs}_\Omega)$
    where $P_0$ on input $x$ and $w$:
      return $\pi \leftarrow_r \mathsf{Prove}_\Omega(x, w)$, if $R(x, w) = 1$
      return $\perp$
    and $P_1$ on input $(x, w)$:
      return $\pi \leftarrow_r \mathsf{SIM}_2(\mathsf{crs}_\Omega, \tau, x)$, if $R(x, w) = 1$
      return $\perp$
  return 1, if $b^* = b$
  return 0
```

Fig. 4: Ω Zero-Knowledge

```
$\mathbf{Exp}^{\mathsf{SimSoundExt}}_{\mathcal{A},\Omega,\mathcal{E},L}(\kappa)$
  $(\mathsf{crs}_\Omega, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa, L)$
  $(x^*, \pi^*) \leftarrow_r \mathcal{A}^{\mathsf{SIM}(\cdot)}(\mathsf{crs}_\Omega)$
  $\mathcal{Q} \leftarrow \emptyset$
    where $\mathsf{SIM}$ on input $x$:
      obtain $\pi \leftarrow_r \mathcal{E}_2(\mathsf{crs}_\Omega, \tau, x)$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(x, \pi)\}$
      return $\pi$
  $w^* \leftarrow_r \mathcal{E}_3(\mathsf{crs}_\Omega, \xi, x^*, \pi^*)$
  return 1, if $\mathsf{Verify}_\Omega(x^*, \pi^*) = 1 \ \wedge \ R(x^*, w^*) = 0 \ \wedge \ (x^*, \pi^*) \notin \mathcal{Q}$
  return 0
```

Fig. 5: Ω Simulation Sound Extractability

Definition 9 (Ω Simulation-Sound Extractability). *A zero-knowledge non-interactive proof system Ω is said to be simulation-sound extractable, if for a fixed language L, for any PPT adversary $\mathcal{A}$, there exists a PPT extractor/simulator $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3)$, such that there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{SimSoundExt}}_{\mathcal{A},\Omega,\mathcal{E},L}(\kappa)\right] = 1 \leq \nu(\kappa)$$

The corresponding experiment is depicted in Figure 5.

Note, $\mathcal{E}_1$, $\mathcal{E}_2$ are required to behave exactly as $(\mathsf{SIM}_1, \mathsf{SIM}_2)$ from the zero-knowledge definition [39], where ξ is the extraction trapdoor.

For the sake of readability, a somewhat informal CS-notation, derived from Camenisch and Stadler [21], is used. For example, the notation $\pi \leftarrow_r \mathsf{Prove}_\Omega\{(g_1) : C = \mathsf{Enc}_\Pi(g_1)\}(\ell)$ denotes the computation of a non-interactive, simulation-sound extractable, zero-knowledge proof of knowledge (NIZKPOK for short) of the plaintext g_1 contained in C (which is assumed to be public), with a non-malleable attached label ℓ. This is also known as a "signature of knowledge" [25]. Sometimes only "verify π" is used for verification of a proof π. It is assumed that the public parameters, and the statement to be proven, are also input to the proof system as the label and are public. This is not make explicit to increase readability.

We also require to strengthen a recent primitive, dubbed policy-based chameleon-hashes, introduced in [30]. Before we introduce it, we need to define what an access structure is.

Definition 10 (Access Structure). *Let $\mathbb{U}$ denote the universe of attributes. A collection $\mathbb{A} \in 2^{\mathbb{U}} \setminus \{\emptyset\}$ of non-empty sets is an access structure on $\mathbb{U}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets. A collection $\mathbb{A} \in 2^{\mathbb{U}} \setminus \{\emptyset\}$ is called monotone if $\forall\ B, C \in \mathbb{A}$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$.*

Definition 11 (Policy-Based Chameleon-Hashes). *A policy-based chameleon-hash* PCH *consists of six algorithms* $(\mathsf{PPGen}_{\mathsf{PCH}}, \mathsf{MKeyGen}_{\mathsf{PCH}}, \mathsf{KGen}_{\mathsf{PCH}}, \mathsf{Hash}_{\mathsf{PCH}}, \mathsf{Verify}_{\mathsf{PCH}}, \mathsf{Adapt}_{\mathsf{PCH}})$ *which are defined as follows [30].*

$\mathsf{PPGen}_{\mathsf{PCH}}$. *On input a security parameter κ,* $\mathsf{PPGen}_{\mathsf{PCH}}$ *outputs the public parameters:*

$$\mathsf{pp}_{\mathsf{PCH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)$$

We assume that $\mathsf{pp}_{\mathsf{PCH}}$ *contains 1^κ and is implicit input to all other algorithms.*

$\mathsf{MKeyGen}_{\mathsf{PCH}}$. *On input of the global parameters* $\mathsf{pp}_{\mathsf{PCH}}$, $\mathsf{MKeyGen}_{\mathsf{PCH}}$ *outputs the master private and public key* $(\mathsf{sk}_{\mathsf{PCH}}, \mathsf{pk}_{\mathsf{PCH}})$ *of the scheme:*

$$(\mathsf{sk}_{\mathsf{PCH}}, \mathsf{pk}_{\mathsf{PCH}}) \leftarrow_r \mathsf{MKeyGen}_{\mathsf{PCH}}(\mathsf{pp}_{\mathsf{PCH}})$$

$\mathsf{KGen}_{\mathsf{PCH}}$. *On input a secret key* sk *and a set of attributes $\mathbb{S} \subseteq \mathbb{U}$, the key generation algorithm outputs a secret key* $\mathsf{sk}_{\mathbb{S}}$*:*

$$\mathsf{sk}_{\mathbb{S}} \leftarrow_r \mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \mathbb{S})$$

$\mathsf{Hash}_{\mathsf{PCH}}$. *On input a public key* pk*, access structure $\mathbb{A} \subseteq 2^{\mathbb{U}}$ and a message m, this algorithm outputs a hash h and some randomness (sometimes referred to as "check value") r:*

$$(h, r) \leftarrow_r \mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$$

$\mathsf{Verify}_{\mathsf{PCH}}$. *On input public key* pk*, a message m, a hash h, and a randomness r, it outputs a bit $b \in \{1, 0\}$.*

$$d \leftarrow \mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, h, r)$$

$\mathsf{Adapt}_{\mathsf{PCH}}$. *On input a secret key* $\mathsf{sk}_{\mathbb{S}}$*, messages m and m', a hash h, and randomness value r, the adaptation algorithm outputs a new randomness r':*

$$r' \leftarrow_r \mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, \mathsf{sk}_{\mathbb{S}}, m, m', h, r)$$

We assume that the $\mathsf{KGen}_{\mathsf{PCH}}$ outputs $\perp$, if $\mathbb{S}$ is not contained in $\mathbb{U}$.

Note, we have added an additional algorithm $\mathsf{PPGen}_{\mathsf{PCH}}$ which outputs some additional global parameters, which was not part of the original description in [30], as we work in a slightly different setting. For correctness, we require that for all $\kappa \in \mathbb{N}$, for all $\mathsf{pp}_{\mathsf{PCH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)$, for all $(\mathsf{sk}_{\mathsf{PCH}}, \mathsf{pk}_{\mathsf{PCH}}) \leftarrow_r \mathsf{MKeyGen}_{\mathsf{PCH}}(\mathsf{pp}_{\mathsf{PCH}})$, for all $\mathbb{A} \subseteq 2^{\mathbb{U}}$, for all $\mathbb{S} \in \mathbb{A}$, for all $\mathsf{sk}_{\mathbb{S}} \leftarrow_r \mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \mathbb{S})$, for all $m \in \mathcal{M}$, for all $(h, r) \leftarrow_r \mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$, for all $m' \in \mathcal{M}$, for all $r' \leftarrow_r \mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, \mathsf{sk}_{\mathbb{S}}, m, m', h, r)$, we have that that $1 = \mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, h, r) = \mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m', h, r')$.

Furthermore, we require the following security properties, where our notion of indistinguishability below is stronger than the one introduced in [30]. We also restate the black-box construction from [30] (with some minor rephrasing and slightly stronger primitives) in Appendix B. The security proof in our stronger model is given in Appendix A.

Full Indistinguishability. Informally, indistinguishability requires that it be intractable to decide whether for a chameleon-hash its randomness is fresh or was created using the adaption algorithm. Full indistinguishability even lets the adversary choose the secret key used in the $\mathsf{HashOrAdapt}$ oracle. The security experiment grants the adversary access to a left-or-right style $\mathsf{HashOrAdapt}$ oracle and requires that the randomnesses r does not reveal whether it was obtained through $\mathsf{Hash}_{\mathsf{PCH}}$ or $\mathsf{Adapt}_{\mathsf{PCH}}$. The messages and secret keys are adaptively chosen by the adversary.

```
$\mathbf{Exp}^{\mathsf{FIndistinguishability}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$
  $\mathsf{pp}_{\mathsf{PCH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)$
  $b \leftarrow_r \{0,1\}$
  $b^* \leftarrow_r \mathcal{A}^{\mathsf{HashOrAdapt}(\cdot,\cdot,\cdot,\cdot,\cdot,b)}(\mathsf{pp}_{\mathsf{PCH}})$
    where $\mathsf{HashOrAdapt}$ on input $\mathsf{pk}_{\mathsf{PCH}}, m, m', \mathsf{sk}_{\mathbb{S}}, \mathbb{A}, b$:
      $(h_0, r_0) \leftarrow_r \mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m', \mathbb{A})$
      $(h_1, r_1) \leftarrow_r \mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$
      $r_1 \leftarrow_r \mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, \mathsf{sk}_{\mathbb{S}}, m, m', h_1, r_1)$
      return $\perp$, if $r_0 = \perp \vee r_1 = \perp$
      return $(h_b, r_b)$
  return 1, if $b = b^*$
  return 0
```

Fig. 6: PCH Full Indistinguishability

Definition 12 (PCH **Full Indistinguishability**). *We say a* PCH *scheme is fully indistinguishable, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\left|\Pr\left[\mathbf{Exp}^{\mathsf{FIndistinguishability}}_{\mathcal{A},\mathsf{PCH}}(\kappa) = 1\right] - \tfrac{1}{2}\right| \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 6.

Insider Collision-Resistance. Insider collision-resistance addresses the requirement that not even insiders who possess secret keys with respect to some attributes can find collisions for hashes which were computed with respect to policies which are not satisfied by their keys (oracle $\mathsf{KGen}'_{\mathsf{PCH}}$). Intuitively, this notion enforces the attribute-based access-control policies, even if the adversary sees collisions for arbitrary attributes (oracles $\mathsf{KGen}''_{\mathsf{PCH}}$ and $\mathsf{Adapt}'_{\mathsf{PCH}}$).

Definition 13 (PCH **Insider Collision-Resistance**). *We say a* PCH *scheme is insider collision-resistant, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{CRIns}}_{\mathcal{A},\mathsf{PCH}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 7.

Uniqueness. We also introduce the new notion of uniqueness for PCHs, which basically requires that it is hard to find different randomness yielding the same hash for an adversarial chosen message and public key.

Definition 14 (PCH **Uniqueness**). *We say a* PCH *scheme is unique, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{Uniqueness}}_{\mathcal{A},\mathsf{PCH}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 8.

Note, we do not require outsider collision-resistance [30].

```
$\mathbf{Exp}^{\mathsf{CRIns}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$
  $\mathsf{pp_{PCH}} \leftarrow_r \mathsf{PPGen_{PCH}}(1^\kappa)$
  $(\mathsf{sk_{PCH}}, \mathsf{pk_{PCH}}) \leftarrow_r \mathsf{MKeyGen_{PCH}}(\mathsf{pp_{PCH}})$
  $\mathcal{S} = \mathcal{H} = \mathcal{Q} \leftarrow \emptyset$
  $i \leftarrow 0$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow_r \mathcal{A}^{\mathsf{KGen'_{PCH}}(\mathsf{sk_{PCH}},\cdot),\mathsf{KGen''_{PCH}}(\mathsf{sk_{PCH}},\cdot),\mathsf{Hash'_{PCH}}(\mathsf{pk_{PCH}},\cdot,\cdot),\mathsf{Adapt'_{PCH}}(\mathsf{pk_{PCH}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk_{PCH}})$
    where $\mathsf{KGen'_{PCH}}$ on input $\mathsf{sk_{PCH}}$, $\mathbb{S}$:
      $\mathsf{sk}_\mathbb{S} \leftarrow_r \mathsf{KGen_{PCH}}(\mathsf{sk}, \mathbb{S})$
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbb{S}\}$
      return $\mathsf{sk}_\mathbb{S}$
    and $\mathsf{KGen''_{PCH}}$ on input $\mathsf{sk_{PCH}}$, $\mathbb{S}$:
      $\mathsf{sk}_\mathbb{S} \leftarrow_r \mathsf{KGen_{PCH}}(\mathsf{sk}, \mathbb{S})$
      $\mathcal{Q} \cup \{(i, \mathsf{sk}_\mathbb{S})\}$
      $i \leftarrow i + 1$
    and $\mathsf{Hash'_{PCH}}$ on input $\mathsf{pk_{PCH}}, m, \mathbb{A}$:
      $(h, r) \leftarrow_r \mathsf{Hash_{PCH}}(\mathsf{pk_{PCH}}, m, \mathbb{A})$
      if $r \neq \bot$, $\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, \mathbb{A}, m)\}$
      return $(h, r)$
    and $\mathsf{Adapt'_{PCH}}$ on input $\mathsf{pk_{PCH}}, m, m', h, r, j$:
      return $\bot$, if $(j, \mathsf{sk}_\mathbb{S}) \notin \mathcal{Q}$ for some $\mathsf{sk}_\mathbb{S}$
      $r' \leftarrow_r \mathsf{Adapt_{PCH}}(\mathsf{pk_{PCH}}, \mathsf{sk}_\mathbb{S}, m, m', h, r)$
      if $r' \neq \bot \ \wedge \ (h, \mathbb{A}, m) \in \mathcal{H}$ for some $\mathbb{A}$, $\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, \mathbb{A}, m')\}$
      return $r'$
  return 1, if
    $\mathsf{Verify_{PCH}}(\mathsf{pk}, m^*, h^*, r^*) = \mathsf{Verify_{PCH}}(\mathsf{pk}, m'^*, h^*, r'^*) = 1 \ \wedge$
    $(h^*, \mathbb{A}, \cdot) \in \mathcal{H}$, for some $\mathbb{A} \ \wedge \ m^* \neq m'^* \ \wedge \ \mathbb{A} \cap \mathcal{S} = \emptyset \ \wedge \ (h^*, \cdot, m^*) \notin \mathcal{H}$
  return 0
```

Fig. 7: PCH Insider Collision-Resistance

```
$\mathbf{Exp}^{\mathsf{Uniqueness}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$
  $\mathsf{pp_{PCH}} \leftarrow_r \mathsf{PPGen_{PCH}}(1^\kappa)$
  $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*) \leftarrow_r \mathcal{A}(\mathsf{pp_{PCH}})$
  return 1, if $\mathsf{Verify_{PCH}}(\mathsf{pk}^*, m^*, h^*, r^*) = \mathsf{Verify_{PCH}}(\mathsf{pk}^*, m^*, h^*, r'^*) = 1 \ \wedge \ r^* \neq r'^*$
  return 0
```

Fig. 8: PCH Uniqueness

3 Our Framework for P3Ss

Additional Notation. We need to introduce some additional notation, to make our representation more compact. Our notation is taken from existing work, making reading more accessible [6, 13, 20]. The variable A contains the set of indices of the modifiable blocks, as well as ℓ denoting the total number of blocks in the message m. We write $\mathsf{A}(m) = 1$, if A is valid w.r.t. m, i.e., A contains a fitting ℓ, i.e., the correct length of m, and the indices of the admissible blocks are actually part of m. For example, let $\mathsf{A} = (\{1, 2, 3, 5\}, 5)$. Then, m must contain five blocks, and all but the fourth can be modified. If we write $m^i \in \mathsf{A}$, we mean that m^i is admissible. We also use m_A for the list of blocks in m which are admissible w.r.t. A. Likewise, we use $m_{!\mathsf{A}}$ for the list of blocks of m which are not admissible w.r.t. to A. Moreover, M is a set containing pairs (i, m'^i) for those blocks that are modified, meaning that m^i is replaced with m'^i. We write $\mathsf{M}(\mathsf{A}) = 1$, if M is valid w.r.t. A, meaning that the indices to be modified are contained in A, i.e., admissible.

Definitional Framework. We now introduce our definitional framework. It is based on existing work [6,13,20]. The main idea is following the line of reasoning of group-signatures. Namely, a designated entity, which we name "the group-manager", following the terminology of group-signatures, generates a key pair for its group. The group-manager can use its secret key to assign secret keys to sanitizers which are identified by their own key pair. In contrast, signers can create signatures for a specific group (and do not require any prior interaction, i.e., knowledge of the group public-key is sufficient which is a major difference to group-signatures), and any sanitizer within that group can then sanitize the generated signatures. Moreover, in contrast to group-signatures, only the *signer* can generate proofs to achieve accountability. These proofs, however, can be verified by anyone. We keep the wording of the algorithms mostly consistent with existing work to ease readability [13].

Definition 15 ($\mathsf{P3S}$)**.** *A sanitizable signature with attribute-based sanitizing* $\mathsf{P3S}$ *consists of the algorithms* $\{\mathsf{ParGen}_{\mathsf{P3S}}, \mathsf{Setup}_{\mathsf{P3S}}, \mathsf{KGenSig}_{\mathsf{P3S}}, \mathsf{KGenSan}_{\mathsf{P3S}}, \mathsf{Sign}_{\mathsf{P3S}}, \mathsf{AddSan}_{\mathsf{P3S}}, \mathsf{Sanitize}_{\mathsf{P3S}}, \mathsf{Verify}_{\mathsf{P3S}}, \mathsf{Proof}_{\mathsf{P3S}}, \mathsf{Judge}_{\mathsf{P3S}}\}$ *such that:*

$\mathsf{ParGen}_{\mathsf{P3S}}$**.** *The algorithm* $\mathsf{ParGen}_{\mathsf{P3S}}$ *generates the public parameters:*

$$\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$$

We assume that $\mathsf{pp}_{\mathsf{P3S}}$ *contains* 1^κ *and is implicit input to all other algorithms.*

$\mathsf{Setup}_{\mathsf{P3S}}$**.** *The algorithm* $\mathsf{Setup}_{\mathsf{P3S}}$ *outputs the global public key* $\mathsf{pk}_{\mathsf{P3S}}$ *of a* $\mathsf{P3S}$*, and some master secret key* $\mathsf{sk}_{\mathsf{P3S}}$*, i.e., it generates the group-manager's key pair:*

$$(\mathsf{sk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}) \leftarrow_r \mathsf{Setup}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$$

$\mathsf{KGenSig}_{\mathsf{P3S}}$**.** *The algorithm* $\mathsf{KGenSig}_{\mathsf{P3S}}$ *generates a key-pair for a signer:*

$$(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSig}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$$

$\mathsf{KGenSan}_{\mathsf{P3S}}$**.** *The algorithm* $\mathsf{KGenSan}_{\mathsf{P3S}}$ *generates a key-pair for a sanitizer:*

$$(\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSan}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$$

$\mathsf{Sign}_{\mathsf{P3S}}$**.** *The algorithm* $\mathsf{Sign}_{\mathsf{P3S}}$ *generates a signature* σ*, on input of a master public key* $\mathsf{pk}_{\mathsf{P3S}}$*, a secret key* $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$*, a message* m*,* A*, and some access structure* $\mathbb{A}$*:*

$$\sigma \leftarrow_r \mathsf{Sign}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, m, \mathsf{A}, \mathbb{A})$$

$\mathsf{AddSan}_{\mathsf{P3S}}$**.** *The algorithm* $\mathsf{AddSan}_{\mathsf{P3S}}$ *allows to the group-manager to generate a secret sanitizing key* $\mathsf{sk}_{\mathbb{S}}$ *for a particular sanitizer, on input of* $\mathsf{sk}_{\mathsf{P3S}}$*, a public key* $\mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}$*, and some set of attributes* $\mathbb{S} \subseteq \mathbb{U}$*:*

$$\mathsf{sk}_{\mathbb{S}} \leftarrow_r \mathsf{AddSan}_{\mathsf{P3S}}(\mathsf{sk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathbb{S})$$

$\mathsf{Verify}_{\mathsf{P3S}}$**.** *The deterministic algorithm* $\mathsf{Verify}_{\mathsf{P3S}}$ *allows to verify a signature* σ *on input of a master public key* $\mathsf{pk}_{\mathsf{P3S}}$*, a signer public key* $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$*, and a message* m*. It outputs a decision* $b \in \{0, 1\}$*:*

$$b \leftarrow \mathsf{Verify}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma, m)$$

$\mathsf{Sanitize}_{\mathsf{P3S}}$**.** *The algorithm* $\mathsf{Sanitize}_{\mathsf{P3S}}$ *allows to derive a new signature on input of a master public key* $\mathsf{pk}_{\mathsf{P3S}}$*, a signer's public key* $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$*, a sanitizer's secret key* $\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}$*, a token* $\mathsf{sk}_{\mathbb{S}}$*, some modification instruction* M*, a message* m*, and a signature* σ*:*

$$(\sigma', m') \leftarrow_r \mathsf{Sanitize}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{sk}_{\mathbb{S}}, m, \sigma, \mathsf{M})$$

$\mathsf{Proof}_{\mathsf{P3S}}$. *The algorithm* $\mathsf{Proof}_{\mathsf{P3S}}$ *allows to generate a proof* π_{P3S} *and some public* pk*, used by the next algorithm, to find the accountable party, on input of a master public key* $\mathsf{pk}_{\mathsf{P3S}}$*, a signer's secret key* $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$*, a signature* σ*, and a message* m*:*

$$(\pi_{\mathsf{P3S}}, \mathsf{pk}) \leftarrow_r \mathsf{Proof}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma, m)$$

$\mathsf{Judge}_{\mathsf{P3S}}$. *The algorithm* $\mathsf{Judge}_{\mathsf{P3S}}$ *allows to verify whether a proof* π_{P3S} *is valid. The inputs are a master public key* $\mathsf{pk}_{\mathsf{P3S}}$*, a signer's public key* $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$*, some other public key* pk*, a proof* π_{P3S}*, a signature* σ*, and a message* m*. It outputs a decision* $b \in \{0, 1\}$*, stating whether* π_{P3S} *is a valid proof that the holder of* pk *is accountable for* σ*:*

$$b \leftarrow_r \mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}, \pi_{\mathsf{P3S}}, \sigma, m)$$

For each $\mathsf{P3S}$ it is required that the correctness properties hold. In particular, it is required that for all $\kappa \in \mathbb{N}$, for all $\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$, for all $(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}_{\mathsf{P3S}}) \leftarrow_r \mathsf{Setup}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$, for all $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSig}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$, for all $\ell \in \mathbb{N}$, for all $m \in \mathcal{M}^\ell$, for all $\mathbb{A} \in 2^{\mathbb{U}}$, for all $\mathsf{A} \in \{\mathsf{A}_i \mid \mathsf{A}_i(m) = 1\}$, for all $\sigma \leftarrow_r \mathsf{Sign}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, m, \mathsf{A}, \mathbb{A})$, we have that $\mathsf{Verify}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma, m) = 1$ and for all $(\pi_{\mathsf{P3S}}, \mathsf{pk}) \leftarrow_r \mathsf{Proof}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma, m)$ we have that $\mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \pi_{\mathsf{P3S}}, \sigma, m) = 1$ and $\mathsf{pk} = \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$. Moreover, we require that for all $(\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSan}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$, for all $\mathbb{S} \in \mathbb{A}$, for all $\mathsf{sk}_{\mathbb{S}} \leftarrow_r \mathsf{AddSan}_{\mathsf{P3S}}(\mathsf{sk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathbb{S})$, for all $\mathsf{M} \in \{\mathsf{M}_i \mid \mathsf{M}_i(\mathsf{A}) = 1\}$, for all $(\sigma', m') \leftarrow_r \mathsf{Sanitize}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{sk}_{\mathbb{S}}, m, \sigma, \mathsf{M})$ we have that $\mathsf{Verify}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma', m') = 1$ and that for all $(\pi'_{\mathsf{P3S}}, \mathsf{pk}') \leftarrow_r \mathsf{Proof}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma', m')$, we have that $\mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}, \pi'_{\mathsf{P3S}}, \sigma', m') = 1$ and $\mathsf{pk}' = \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}$.

Security Definitions. We now introduce our security definitions. To increase readability, we keep the naming close to the already existing definitions for standard 3Ss [13]. However, due to the increased expressiveness of our new primitive, this is not always possible. Namely, we require new unforgeability and privacy definitions not considered before.

Unforgeability. The property of unforgeability prohibits that an adversary, which is not a signer, or the entity holding $\mathsf{sk}_{\mathsf{P3S}}$, i.e., the group-manager, can generate any validating signature which verifies for honestly generated keys. This also includes messages for which the adversary does not hold enough attributes for, even if it sees sanitizations of such signatures. We define it in such a way that $(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}_{\mathsf{P3S}})$, and $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}})$, are generated honestly. The adversary gets access to the following oracles: (1) $\mathsf{Sign}'_{\mathsf{P3S}}$ (where it can even use different $\mathsf{pk}_{\mathsf{P3S}}$s, which models the case that secret signing keys can be re-used across multiple "groups"), (2) GetSan which generates a new sanitizer (tracked by $\mathcal{S}$), (3) $\mathsf{AddSan}'_{\mathsf{P3S}}$ which allows to decide which attributes a given sanitizer holds (tracked by $\mathcal{R}$), (4) $\mathsf{Sanitize}'_{\mathsf{P3S}}$ which allows sanitizing signatures for an honest sanitizer (generated by GetSan) for the challenge group, and (5) $\mathsf{Sanitize}''_{\mathsf{P3S}}$ which allows sanitizing for signatures from any other group (i.e., where the adversary is the group manager). The adversary wins, if it can generate a valid signature for the defined group which has never been output by either $\mathsf{Sign}'_{\mathsf{P3S}}$ or $\mathsf{Sanitize}'_{\mathsf{P3S}}$ (tracked by the set $\mathcal{M}$; Note, this set may be exponential in size, but membership is trivial to decide), and the adversary $\mathcal{A}$ does not hold enough attributes itself.

Definition 16 ($\mathsf{P3S}$ Unforgeability). *We say a* $\mathsf{P3S}$ *scheme is unforgeable, if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* ν *such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{Unforgeability}}_{\mathcal{A},\mathsf{P3S}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 9.

Immutability. The above unforgeability definitions assume that the holder of $\mathsf{sk}_{\mathsf{P3S}}$ is honest. If this is not the case, however, the adversary can generate its own key pair for a sanitizer and can generate $\mathsf{sk}_{\mathbb{S}}$ for any attribute-set it likes. Still, in such a case, we want to prohibit that an adversary can generate any signatures

```
$\mathbf{Exp}^{\mathsf{Unforgeability}}_{\mathcal{A},\mathsf{P3S}}(\kappa)$
  $\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$
  $(\mathsf{sk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}) \leftarrow_r \mathsf{Setup}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$
  $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSig}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$
  $\mathcal{Q} = \mathcal{S} = \mathcal{R} = \mathcal{M} = \mathcal{Z} \leftarrow \emptyset$
  $i \leftarrow 0$
  $(m^*, \sigma^*) \leftarrow_r \mathcal{A}^{\mathsf{Sign}'_{\mathsf{P3S}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot,\cdot),\mathsf{GetSan}(),\mathsf{AddSan}'_{\mathsf{P3S}}(\mathsf{sk}_{\mathsf{P3S}},\cdot,\cdot),\mathsf{Sanitize}'_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}},\cdot,\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{Sanitize}''_{\mathsf{P3S}}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{Proof}_{\mathsf{P3S}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot)}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}})$
    where $\mathsf{Sign}'_{\mathsf{P3S}}$ on input $\mathsf{pk}'_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $m$, $\mathsf{A}$, $\mathbb{A}$:
      $\sigma \leftarrow_r \mathsf{Sign}_{\mathsf{P3S}}(\mathsf{pk}'_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, m, \mathsf{A}, \mathbb{A})$
      if $\mathsf{pk}'_{\mathsf{P3S}} = \mathsf{pk}_{\mathsf{P3S}} \ \wedge \ \sigma \neq \perp$:
        $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\sigma, m, \mathbb{A}, \mathsf{A})\}$
        if $\mathbb{A} \in \mathcal{R}$, $\mathcal{M} \leftarrow \mathcal{M} \cup \{\mathsf{M}(m) \mid \mathsf{M}(\mathsf{A}) = 1\}$
      return $\sigma$
    and $\mathsf{GetSan}$:
      $(\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSan}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}})\}$
      return $\mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}$
    and $\mathsf{AddSan}'_{\mathsf{P3S}}$ on input $\mathsf{sk}_{\mathsf{P3S}}$, $\mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}$, $\mathbb{S}$
      if $\neg\exists(\cdot, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}) \in \mathcal{S}$:
        $\mathsf{sk}_{\mathbb{S}} \leftarrow_r \mathsf{AddSan}_{\mathsf{P3S}}(\mathsf{sk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathbb{S})$
        return $\perp$, if $\mathsf{sk}_{\mathbb{S}} = \perp$
        $\mathcal{R} \leftarrow \mathcal{R} \cup \{\mathbb{S}\}$
        for all $(\sigma_i, m_i, \mathbb{A}_i, \mathsf{A}_i) \in \mathcal{Q}$, where $\mathbb{S} \in \mathbb{A}_i$, $\mathcal{M} \cup \{\mathsf{M}(m_i) \mid \mathsf{M}(\mathsf{A}_i) = 1\}$
        return $\mathsf{sk}_{\mathbb{S}}$
      $\mathsf{sk}_{\mathbb{S}} \leftarrow_r \mathsf{AddSan}_{\mathsf{P3S}}(\mathsf{sk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathbb{S})$
      $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{(i, \mathsf{sk}_{\mathbb{S}})\}$
      $i \leftarrow i + 1$
    and $\mathsf{Sanitize}'_{\mathsf{P3S}}$ on input $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $\mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}$, $j$, $m$, $\sigma$, $\mathsf{M}$:
      return $\perp$, if $\neg\exists(\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}) \in \mathcal{S}$ for some $\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}$
      $(\sigma', m') \leftarrow_r \mathsf{Sanitize}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{sk}_{\mathbb{S}}, m, \sigma, \mathsf{M})$, where $\mathsf{sk}_{\mathbb{S}}$ is taken from $(j, \mathsf{sk}_{\mathbb{S}}) \in \mathcal{Z}$
      if $\mathsf{pk}'_{\mathsf{P3S}} = \mathsf{pk}_{\mathsf{P3S}} \wedge \sigma' \neq \perp$:
        $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\sigma', m', \perp, \perp)\}$
      return $\sigma'$
    and $\mathsf{Sanitize}''_{\mathsf{P3S}}$ on input $\mathsf{pk}'_{\mathsf{P3S}}$, $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $\mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}$, $\mathsf{sk}_{\mathbb{S}}$, $m$, $\sigma$, $\mathsf{M}$:
      return $\perp$, if $\neg\exists(\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}) \in \mathcal{S} \ \vee \ \mathsf{pk}'_{\mathsf{P3S}} = \mathsf{pk}_{\mathsf{P3S}}$
      $(\sigma', m') \leftarrow_r \mathsf{Sanitize}_{\mathsf{P3S}}(\mathsf{pk}'_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{sk}_{\mathbb{S}}, m, \sigma, \mathsf{M})$
      return $\sigma'$
  return 0, if $\mathsf{Verify}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma^*, m^*) = 0 \ \vee \ m^* \in \mathcal{M}$
      return 1, if $(\sigma^*, m^*, \cdot, \cdot) \notin \mathcal{Q}$
  return 0
```

Fig. 9: P3S Unforgeability

which are outside the span the honest signer has endorsed for *any* combination of attributes. This is captured within the immutability definition — if a block is marked as non-admissible by a signer, no-one must be able to change this block. This also includes that an adversary must not be able to redact or append a block. Clearly, we cannot limit the adversary to change admissible blocks, as it can grant sanitizing rights to itself.

This is modeled in such a way that the challenger draws $\mathsf{pp}_{\mathsf{P3S}}$ honestly, along with a key-pair for the signer. The adversary only receives $\mathsf{pp}_{\mathsf{P3S}}$ and $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$. Then, the adversary gains adaptive access to signing-oracle (where the adversary can choose $\mathsf{pk}_{\mathsf{P3S}}$, m, A, $\mathbb{A}$, but not $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$), and access to a proof-oracle. We keep a set $\mathcal{M}$ which contains all possible messages which can "legally" be derived by the adversary (bound to $\mathsf{pk}_{\mathsf{P3S}}$, also chosen by the adversary, and tracked by $\mathcal{M}$; Again, this set may be exponential in size, but membership is trivial to decide). If, and only if, the adversary finds a valid signature σ^* w.r.t. $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$ and pk^*, which could never been derived from any input, it wins.

```
$\mathbf{Exp}^{\mathsf{Immutability}}_{\mathcal{A},\mathsf{P3S}}(\kappa)$
  $\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$
  $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSig}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$
  $\mathcal{M} \leftarrow \emptyset$
  $(\mathsf{pk}^*, \sigma^*, m^*) \leftarrow_r \mathcal{A}^{\mathsf{Sign}'_{\mathsf{P3S}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot,\cdot),\mathsf{Proof}_{\mathsf{P3S}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot)}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}})$
    where $\mathsf{Sign}'_{\mathsf{P3S}}$ on input $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $m$, $\mathsf{A}$, $\mathbb{A}$:
      $\sigma \leftarrow_r \mathsf{Sign}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, m, \mathsf{A}, \mathbb{A})$
      return $\perp$, if $\sigma = \perp$
      $\mathcal{M} \cup \{(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{M}(m)) \mid \mathsf{M}(\mathsf{A}) = 1\}$
      return $\sigma$
  return 1, if:
    $\mathsf{Verify}_{\mathsf{P3S}}(\mathsf{pk}^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma^*, m^*) = 1 \ \wedge \ (\mathsf{pk}^*, m^*) \notin \mathcal{M}$
  return 0
```

Fig. 10: $\mathsf{P3S}$ Immutability

Definition 17 ($\mathsf{P3S}$ **Immutability**). *We say a* $\mathsf{P3S}$ *scheme is immutable, if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* ν *such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{Immutability}}_{\mathcal{A},\mathsf{P3S}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 10.

Privacy. Privacy prohibits that an adversary can derive any useful information from a sanitized signature. We define a very strong version, where all values can be generated by the adversary, making our definition even stronger than existing ones [27,33].

In more detail, the challenger draws a bit $b \leftarrow_r \{0,1\}$, while the parameters $\mathsf{pp}_{\mathsf{P3S}}$ are generated honestly. The adversary gains access to a $\mathsf{LoRSanit}$-oracle, where it can input $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}$, $\mathbb{A}$, m_0, m_1, M_0, M_1, A, and $\mathsf{sk}_{\mathbb{S}}$ (b is input by the challenger). The oracle then signs m_b with A and $\mathbb{A}$. Then, the resulting signature is sanitized to $\mathsf{M}_b(m_b)$, while $\mathsf{M}_0(m_0) = \mathsf{M}_1(m_1)$ must hold to prevent trivial attacks. The goal of the adversary is to guess the bit b.

We stress that this definition seems to be overly strong. However, it also preserves privacy in case of bad randomness at key generation, completely leaked keys, and even corrupt group-managers.

Definition 18 ($\mathsf{P3S}$ **Privacy**). *We say a* $\mathsf{P3S}$ *scheme is private, if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* ν *such that:*

$$\left|\Pr\left[\mathbf{Exp}^{\mathsf{Privacy}}_{\mathcal{A},\mathsf{P3S}}(\kappa) = 1\right] - \tfrac{1}{2}\right| \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 11.

Transparency. Transparency prohibits that an adversary can decide whether a signature is fresh or the result of a sanitization. As for privacy, we define a very strong version, where all values, but the signer's key pair $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}})$, can be generated by the adversary, making our definition even stronger than existing ones [27,33,45]. The reason why the signer's key pair must be generated honestly is that the signer can always pinpoint the accountable party due to correctness.

```
$\mathbf{Exp}^{\mathsf{Privacy}}_{\mathcal{A},\mathsf{P3S}}(\kappa)$
  $\mathsf{pp_{P3S}} \leftarrow_r \mathsf{ParGen_{P3S}}(1^\kappa)$
  $b \leftarrow_r \{0,1\}$
  $b^* \leftarrow_r \mathcal{A}^{\mathsf{LoRSanit}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,b)}(\mathsf{pp_{P3S}})$
    where $\mathsf{LoRSanit}$ on input of $\mathsf{pk_{P3S}}$, $\mathsf{sk^{Sig}_{P3S}}$, $\mathsf{sk^{San}_{P3S}}$, $\mathbb{A}$, $m_0$, $m_1$, $\mathsf{M}_0$, $\mathsf{M}_1$, $\mathsf{A}$, $\mathsf{sk}_\mathbb{S}$, $b$:
      $\sigma \leftarrow_r \mathsf{Sign_{P3S}}(\mathsf{pk_{P3S}}, \mathsf{sk^{Sig}_{P3S}}, m_b, \mathsf{A}, \mathbb{A})$
      for $b \in \{0,1\}$, $(\sigma'_b, \cdot) \leftarrow_r \mathsf{Sanitize_{P3S}}(\mathsf{pk_{P3S}}, \mathsf{pk^{Sig}_{P3S}}, \mathsf{sk^{San}_{P3S}}, \mathsf{sk}_\mathbb{S}, m_b, \sigma, \mathsf{M}_b)$
      return $\perp$, if $\sigma'_0 = \perp \ \vee \ \sigma'_1 = \perp \ \vee \ \mathsf{A}(m_0) = 0 \ \vee \ \mathsf{A}(m_1) = 0 \ \vee \ \mathsf{M}_0(m_0) \neq \mathsf{M}_1(m_1)$
      return $\sigma'_b$
  return 1, if $b = b^*$
  return 0
```

Fig. 11: P3S Privacy

In more detail, the challenger draws a bit $b \leftarrow_r \{0,1\}$, while the parameters $\mathsf{pp_{P3S}}$ and the signer's key pair $(\mathsf{sk^{Sig}_{P3S}}, \mathsf{pk^{Sig}_{P3S}})$ are generated honestly. The adversary gains access to three oracles: $\mathsf{Sign_{P3S}}$, $\mathsf{SignOrSanit}$, and $\mathsf{Proof'_{P3S}}$. The $\mathsf{Sign_{P3S}}$-oracle allows the adversary to generate new signatures; the only fixed input is $\mathsf{sk^{Sig}_{P3S}}$. The $\mathsf{SignOrSanit}$-oracle is the challenge oracle. It allows the adversary $\mathcal{A}$ to input $\mathsf{pk_{P3S}}$, $\mathsf{sk^{San}_{P3S}}$, $\mathbb{A}$, m, M, A, and $\mathsf{sk}_\mathbb{S}$ (b and $\mathsf{sk^{Sig}_{P3S}}$ are input by the challenger). The oracle then signs m with A and $\mathbb{A}$. Then, the resulting signature is sanitized to $\mathsf{M}(m)$. If $b = 1$, however, a fresh signature on $\mathsf{M}(m)$ is generated. The resulting signature is returned to the adversary. However, we also log the signatures generated by this oracle in a list $\mathcal{Q}$. The list $\mathcal{Q}$ is required to prohibit that the adversary $\mathcal{A}$ wants to generate a proof using the $\mathsf{Proof'_{P3S}}$-oracle with signatures generated by the $\mathsf{SignOrSanit}$-oracle, which directly returns the accountable party. Thus, the adversary can only input $\mathsf{pk_{P3S}}$, $\mathsf{sk^{Sig}_{P3S}}$, σ, m for which $(\mathsf{pk_{P3S}}, \sigma, m)$ was never input/output to the $\mathsf{SignOrSanit}$-oracle. The goal of the adversary is to guess the bit b.

We stress that this definition also seems to be overly strong. However, it also preserves transparency in case of bad randomness at key generation, leaked keys, and even corrupt group-managers.

Definition 19 (P3S Transparency). *We say a* P3S *scheme is transparent, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\left|\Pr\left[\mathbf{Exp}^{\mathsf{Transparency}}_{\mathcal{A},\mathsf{P3S}}(\kappa) = 1\right] - \tfrac{1}{2}\right| \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 12.

Pseudonymity. Pseudonymity prohibits that an adversary can decide which sanitizer actually is responsible for a given signature, if it does not have access to $\mathsf{sk^{Sig}_{P3S}}$. This is related to the anonymity of group signatures [26]. We formalize it in the following way. The challenger draws a bit $b \leftarrow_r \{0,1\}$, generates the public parameters $\mathsf{pp_{P3S}}$ and the signer's key pair $(\mathsf{sk^{Sig}_{P3S}}, \mathsf{pk^{Sig}_{P3S}})$ honestly. The adversary gains access to three oracles: $\mathsf{Sign_{P3S}}$, $\mathsf{LoRSanit}$, and $\mathsf{Proof'_{P3S}}$. The $\mathsf{Sign_{P3S}}$-oracle allows the adversary to generate new signatures; the only fixed input is $\mathsf{sk^{Sig}_{P3S}}$. The $\mathsf{LoRSanit}$-oracle is the challenge oracle. It allows the adversary $\mathcal{A}$ to input $\mathsf{pk_{P3S}}$, $\mathsf{pk^{Sig}_{P3S}}$, $\mathsf{sk^{San}_{P3S,0}}$, $\mathsf{sk^{San}_{P3S,1}}$, $\mathsf{sk}_{\mathbb{S}0}$, $\mathsf{sk}_{\mathbb{S}1}$, m, and σ (b and $\mathsf{sk^{Sig}_{P3S}}$ are input by the challenger). The oracle then signs m with A and $\mathbb{A}$. Then, the resulting signature is sanitized to $\mathsf{M}(m)$, using keys $\mathsf{sk^{San}_{P3S,\mathit{b}}}$ and $\mathsf{sk}_{\mathbb{S},b}$. The resulting signature is given to the adversary. As done for transparency, we also log the signatures generated by this oracle in a list $\mathcal{Q}$. The list $\mathcal{Q}$ is required to prohibit that the adversary $\mathcal{A}$ wants to generate a proof using the $\mathsf{Proof'_{P3S}}$-oracle with signatures generated by the $\mathsf{LoRSanit}$-oracle, which clearly contradicts pseudonymity. Thus, the adversary can only input $\mathsf{pk_{P3S}}$, $\mathsf{sk^{Sig}_{P3S}}$, σ, m for which $(\mathsf{pk_{P3S}}, \sigma, m)$ was never input/output to the $\mathsf{LoRSanit}$-oracle. The goal of the adversary is to guess the bit b.

```
$\mathbf{Exp}^{\mathsf{Transparency}}_{\mathcal{A},\mathsf{P3S}}(\kappa)$
  $\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$
  $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSig}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$
  $b \leftarrow_r \{0,1\}$
  $\mathcal{Q} \leftarrow \emptyset$
  $b^* \leftarrow_r \mathcal{A}^{\mathsf{Sign}_{\mathsf{P3S}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot,\cdot),\mathsf{SignOrSanit}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,b),\mathsf{Proof}'_{\mathsf{P3S}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot)}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}})$
    where $\mathsf{SignOrSanit}$ on input of $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}$, $\mathbb{A}$, $m$, $\mathsf{M}$, $\mathsf{A}$, $\mathsf{sk}_{\mathbb{S}}$, $b$:
      $\sigma \leftarrow_r \mathsf{Sign}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, m, \mathsf{A}, \mathbb{A})$
      $(\sigma', m') \leftarrow_r \mathsf{Sanitize}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{sk}_{\mathbb{S}}, m, \sigma, \mathsf{M})$
      if $b = 1$:
        $\sigma' \leftarrow_r \mathsf{Sign}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, m', \mathsf{A}, \mathbb{A})$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}_{\mathsf{P3S}}, \sigma', m')\}$
      return $\sigma'$
    and $\mathsf{Proof}'_{\mathsf{P3S}}$ on input of $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $\sigma$, $m$:
      return $\perp$, if $(\mathsf{pk}_{\mathsf{P3S}}, \sigma, m) \in \mathcal{Q}$
      return $\mathsf{Proof}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma, m)$
  return 1, if $b = b^*$
  return 0
```

Fig. 12: P3S Transparency

Again, we stress that this definition also seems to be overly strong. However, as also done for group signatures, secrets keys may leak over time. This definition protects even against bad randomness at key generation.

```
$\mathbf{Exp}^{\mathsf{Pseudonymity}}_{\mathcal{A},\mathsf{P3S}}(\kappa)$
  $\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$
  $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSig}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $b \leftarrow_r \{0,1\}$
  $b^* \leftarrow_r \mathcal{A}^{\mathsf{Sign}_{\mathsf{P3S}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot,\cdot),\mathsf{Proof}'_{\mathsf{P3S}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot),\mathsf{LoRSanit}(\cdot,\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,b)}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}})$
    where $\mathsf{Proof}'_{\mathsf{P3S}}$ on input of $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $\sigma$, $m$:
      return $\perp$, if $(\mathsf{pk}_{\mathsf{P3S}}, \sigma', m') \in \mathcal{Q}$
      return $\mathsf{Proof}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma, m)$
    and $\mathsf{LoRSanit}$ on input of $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S},0}$, $\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S},1}$, $\mathsf{sk}_{\mathbb{S}0}$, $\mathsf{sk}_{\mathbb{S}1}$, $m$, $\sigma$, $b$:
      for $b \in \{0,1\}$, $(\sigma'_b, m'_b) \leftarrow_r \mathsf{Sanitize}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S},b}, \mathsf{sk}_{\mathbb{S},b}, m, \sigma, \mathsf{M})$
      return $\perp$, if $\sigma'_0 = \perp \vee \sigma'_1 = \perp$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}_{\mathsf{P3S}}, \sigma'_b, m'_b)\}$
      return $\sigma'_b$
  return 1, if $b = b^*$
  return 0
```

Fig. 13: P3S Pseudonymity

Definition 20 (P3S **Pseudonymity**). *We say a* P3S *scheme is pseudonymous, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\left|\Pr\left[\mathbf{Exp}^{\mathsf{Pseudonymity}}_{\mathcal{A},\mathsf{P3S}}(\kappa) = 1\right] - \frac{1}{2}\right| \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 13.

Signer-Accountability. Signer-accountability prohibits that an adversary can generate a bogus proof that makes $\mathsf{Judge}_{\mathsf{P3S}}$ decide that a sanitizer is responsible for a given signature/message pair (m^*, σ^*), but that sanitizer has never generated this pair. This is even true, if the adversary can generate the signer's key pair, the global group key pair, while receiving full adaptive access to a sanitization-oracle.

```
$\mathbf{Exp}_{\mathcal{A},\mathsf{P3S}}^{\mathsf{Signer\text{-}Accountability}}(\kappa)$
  $\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$
  $(\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{San}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{San}}) \leftarrow_r \mathsf{KGenSan}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$
  $b \leftarrow_r \{0,1\}$
  $\mathcal{Q} \leftarrow \emptyset$
  $(\mathsf{pk}_0^*, \mathsf{pk}_1^*, \sigma^*, m^*, \pi^*) \leftarrow_r \mathcal{A}^{\mathsf{Sanitize}'_{\mathsf{P3S}}(\cdot,\cdot,\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{San}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{P3S}}^{\mathsf{San}})$
    where $\mathsf{Sanitize}'_{\mathsf{P3S}}$ on input of $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}$, $\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{San}}$, $\mathsf{sk}_{\mathbb{S}}$, $m$, $\sigma$, $\mathsf{M}$:
      $(\sigma', m') \leftarrow_r \mathsf{Sanitize}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{P3S}}^{\mathsf{San}}, \mathsf{sk}_{\mathbb{S}}, m, \sigma, \mathsf{M})$
      if $\sigma \neq \perp$, $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, \sigma', m')\}$
      return $\sigma'$
  return 1, if $\mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}_0^*, \mathsf{pk}_1^*, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{San}}, \pi^*, \sigma^*, m^*) = 1 \ \wedge \ (\mathsf{pk}_0^*, \mathsf{pk}_1^*, \sigma^*, m^*) \notin \mathcal{Q}$
  return 0
```

Fig. 14: P3S Signer-Accountability

Definition 21 **(P3S Signer-Accountability).** *We say a* P3S *scheme is signer-accountable, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}_{\mathcal{A},\mathsf{P3S}}^{\mathsf{Signer\text{-}Accountability}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 14.

Sanitizer-Accountability. Sanitizer-accountability prohibits that an adversary can generate a bogus signature/message pair (m^*, σ^*) that makes $\mathsf{Proof}_{\mathsf{P3S}}$ outputs a (honestly generated) generated proof π_{P3S} which points to the signer, but (m^*, σ^*) has never been generated by the signer. This is even true, if the adversary can generate all sanitizers key pairs, while receiving full adaptive access to a signing-oracle and a proof-oracle.

Definition 22 **(P3S Sanitizer-Accountability).** *We say a* P3S *scheme is sanitizer-accountable, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}_{\mathcal{A},\mathsf{P3S}}^{\mathsf{Sanitizer\text{-}Accountability}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 15.

Proof-Soundness. Proof-Soundness essentially only handles the case that a signature σ can only be opened in an unambiguous way. Thus, the adversary's goal is to output two proofs which "prove" different statements for the same signature/message pair. It is related to the property of opening-soundness introduced by Sakai et al. [48] for group signatures.

Definition 23 **(P3S Proof-Soundness).** *We say a* P3S *scheme is proof-sound, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}_{\mathcal{A},\mathsf{P3S}}^{\mathsf{Proof\text{-}Soundness}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 16.

```
$\mathbf{Exp}^{\mathsf{Sanitizer\text{-}Accountability}}_{\mathcal{A},\mathsf{P3S}}(\kappa)$
  $\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$
  $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSig}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$
  $b \leftarrow_r \{0,1\}$
  $\mathcal{Q} \leftarrow \emptyset$
  $(\mathsf{pk}^*, \sigma^*, m^*, \pi^*) \leftarrow_r \mathcal{A}^{\mathsf{Sign}'_{\mathsf{P3S}}(\cdot, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \cdot, \cdot, \cdot), \mathsf{Proof}_{\mathsf{P3S}}(\cdot, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \cdot, \cdot)}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}})$
    where $\mathsf{Sign}'_{\mathsf{P3S}}$ on input of $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $m$, $\mathsf{A}$, $\mathbb{A}$:
      $\sigma \leftarrow_r \mathsf{Sign}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, m, \mathsf{A}, \mathbb{A})$
      if $\sigma \neq \perp$, $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}_{\mathsf{P3S}}, \sigma', m')\}$
      return $\sigma'$
  $(\pi_{\mathsf{P3S}}, \mathsf{pk}) \leftarrow_r \mathsf{Proof}_{\mathsf{P3S}}(\mathsf{pk}^*, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma^*, m^*)$
  return 1, if $\mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \pi_{\mathsf{P3S}}, \sigma^*, m^*) = 1 \ \wedge \ (\mathsf{pk}^*, \sigma^*, m^*) \notin \mathcal{Q}$
  return 0
```

Fig. 15: P3S Sanitizer-Accountability

```
$\mathbf{Exp}^{\mathsf{Proof\text{-}Soundness}}_{\mathcal{A},\mathsf{P3S}}(\kappa)$
  $\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$
  $((\mathsf{pk}^*_i)_{0 \le i \le 5}, \sigma^*, m^*, \pi^*_0, \pi^*_1) \leftarrow_r \mathcal{A}(\mathsf{pp}_{\mathsf{P3S}})$
  return 1, if $\mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}^*_0, \mathsf{pk}^*_1, \mathsf{pk}^*_2, \pi^*_0, \sigma^*, m^*) = \mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}^*_3, \mathsf{pk}^*_4, \mathsf{pk}^*_5, \pi^*_1, \sigma^*, m^*) = 1 \wedge$
    $(\mathsf{pk}^*_0, \mathsf{pk}^*_1, \mathsf{pk}^*_2) \neq (\mathsf{pk}^*_3, \mathsf{pk}^*_4, \mathsf{pk}^*_5)$
  return 0
```

Fig. 16: P3S Proof-Soundness

Traceability. Traceability requires that an adversary cannot generate a signature which cannot be opened, i.e., it can be seen as the "dual" to proof-soundness. In more detail, the adversary's goal is to generate a verifying signature for which an honest signer cannot generate $(\pi_{\mathsf{P3S}}, \mathsf{pk})$ for which $\mathsf{Judge}_{\mathsf{P3S}}$ outputs correct.

```
$\mathbf{Exp}^{\mathsf{Traceability}}_{\mathcal{A},\mathsf{P3S}}(\kappa)$
  $\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$
  $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSig}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$
  $(\mathsf{pk}^*, \sigma^*, m^*) \leftarrow_r \mathcal{A}^{\mathsf{Sign}_{\mathsf{P3S}}(\cdot, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \cdot, \cdot, \cdot), \mathsf{Proof}_{\mathsf{P3S}}(\cdot, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \cdot, \cdot)}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}})$
  return 0, if $\mathsf{Verify}_{\mathsf{P3S}}(\mathsf{pk}^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma^*, m^*) = 0$
  $(\pi_{\mathsf{P3S}}, \mathsf{pk}) \leftarrow_r \mathsf{Proof}_{\mathsf{P3S}}(\mathsf{pk}^*, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma^*, m^*)$
  return 1, if $\mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}, \pi_{\mathsf{P3S}}, \sigma^*, m^*) = 0$
  return 0
```

Fig. 17: P3S Traceability

Definition 24 ($\mathsf{P3S}$ **Traceability).** *We say a* $\mathsf{P3S}$ *scheme is traceable, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{Traceability}}_{\mathcal{A},\mathsf{P3S}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 17.

Relationship of Properties. All properties are independent of each other. The full theorems and proofs are given in Appendix D.

4 Construction

In this section we present our $\mathsf{P3S}$ construction. The key ingredients are our strengthened version of a policy-based chameleon-hash, a labeled simulation-sound extractable non-interactive zero-knowledge proof of knowledge system (NIZKPOK for short), a one-way function as well as a key-verifiable IND-CPA secure public key encryption scheme[2] and an eUNF-CMA-secure signature scheme. The intuition behind our construction, given in Construction 1, is as follows.

The global parameters of the scheme are a one-way function f, the CRS of the NIZKPOK, and the parameters for the encryption scheme, the signature scheme and the policy-based chameleon hash. The group setup generates the keys of the policy-based chameleon-hash, and a key pair of the signature scheme. The signer generates a signature key pair and publishes the public key together with an image y_1 of a random pre-image x_1 of the OWF f. The sanitizer chooses a random pre-image x_2 of the OWF as secret key and as public key $y_2 = f(x_2)$. If a sanitizers joins a group, i.e., obtains secret keys for a set of attributes $\mathbb{S}$, the group manager signs the sanitizer's public key and additionally issues a secret key for the PCH for attributes $\mathbb{S}$.

For signing, the signer hashes the message using the PCH and signs the hash (along with some additional information). Moreover, it computes a NIZKPOK for the relation R in (1) (using as label ℓ some additional information like the admissible changes). Sanitizing amounts to computing a collision for the PCH hash, updating the respective message blocks, and again attaching a NIZKPOK for relation R. Verification is straightforward.

Let us now briefly discuss the NP relation R which is used within signing and sanitizing to force the signer or the sanitizer to commit to having performed the action. Intuitively, when determining whether a signer or sanitizer has performed the action, the $\mathsf{Proof}_{\mathsf{P3S}}$ algorithm (having access to the group secret key) can simply decrypt c:

$$\begin{aligned}((y_1, c, y_2, \mathsf{pk}_\Pi, \mathsf{pk}_\Sigma),\ (x, r, \sigma_{\mathsf{sk}_\mathbb{S}})) \in R \iff &(y_1 = f(x) \ \wedge\ c = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_1; r)) \\ &\vee\ (y_2 = f(x) \ \wedge\ c = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_2; r) \ \wedge\ \mathsf{Verf}_\Sigma(\mathsf{pk}_\Sigma, y_2, \sigma_{\mathsf{sk}_\mathbb{S}}) = 1).\end{aligned} \quad (1)$$

We note that that it is tempting to think that the weaker notion of witness indistinguishability is sufficient for our construction, but it turns out that one requires zero-knowledge. Moreover, we stress that due to the underlying construction paradigm, we do not consider the strong privacy notion of unlinkability [15], i.e., that sanitized signatures cannot be linked to its origin, which seems to be very hard to achieve with the current construction paradigm. Formally, for our construction, we can show the following:

Theorem 1. *If f is a one-way function, Π is IND-CPA-secure and key-verifiable, Σ is eUNF-CMA secure, Ω is zero-knowledge and simulation-sound extractable, while* PCH *is fully indistinguishable, insider collision-resistant, and unique, the construction of a* $\mathsf{P3S}$ *given in Construction 1 is unforgeable, immutable, private, transparent, pseudonymous, signer-accountable, sanitizer-accountable, proof-sound, and traceable. Likewise, the construction is correct, if the underlying primitives are correct (and sound, resp.).*

[2] Although key-verifiability is no property often explicitly used within IND-CPA encryption schemes, yet common schemes such as ElGamal are key-verifiable. See App. B.

$\underline{\mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)}$: On input a security parameter κ, let $\mathsf{pp}_\Pi \leftarrow_r \mathsf{PPGen}_\Pi(1^\kappa)$, $\mathsf{crs}_\Omega \leftarrow_r \mathsf{PPGen}_\Omega(1^\kappa, L)$.[a] Finally, choose a one-way function f, let $\mathsf{pp}_\Sigma \leftarrow_r \mathsf{PPGen}_\Sigma(1^\kappa)$, and $\mathsf{pp}_{\mathsf{PCH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)$. Return $\mathsf{pp}_{\mathsf{P3S}} \leftarrow (\mathsf{crs}_\Omega, \mathsf{pp}_\Pi, \mathsf{pp}_\Sigma, \mathsf{pp}_{\mathsf{PCH}}, f)$.

$\underline{\mathsf{Setup}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})}$: Let $(\mathsf{sk}_{\mathsf{PCH}}, \mathsf{pk}_{\mathsf{PCH}}) \leftarrow_r \mathsf{MKeyGen}_{\mathsf{PCH}}(\mathsf{pp}_{\mathsf{PCH}})$ and $(\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma) \leftarrow_r \mathsf{KGen}_\Sigma(\mathsf{pp}_\Sigma)$. Return $(\mathsf{sk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}) \leftarrow ((\mathsf{sk}_{\mathsf{PCH}}, \mathsf{sk}_\Sigma), (\mathsf{pk}_{\mathsf{PCH}}, \mathsf{pk}_\Sigma))$.

$\underline{\mathsf{KGenSig}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})}$: Draw $x_1 \leftarrow_r D_f$, $(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) \leftarrow_r \mathsf{KGen}_\Pi(\mathsf{pp}_\Pi)$, let $y_1 \leftarrow f(x_1)$ and $(\mathsf{sk}'_\Sigma, \mathsf{pk}'_\Sigma) \leftarrow_r \mathsf{KGen}_\Sigma(\mathsf{pp}_\Sigma)$.
Return $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}) \leftarrow ((x_1, \mathsf{sk}'_\Sigma, \mathsf{sk}_\Pi), (y_1, \mathsf{pk}'_\Sigma, \mathsf{pk}_\Pi))$.

$\underline{\mathsf{KGenSan}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})}$: Draw $x_2 \leftarrow_r D_f$. Let $y_2 \leftarrow f(x_2)$. Return (x_2, y_2).

$\underline{\mathsf{Sign}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, m, \mathsf{A}, \mathbb{A})}$: If $\mathbb{A} = \emptyset$, return $\perp$. Let $(h, r) \leftarrow_r \mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$, $\sigma_m \leftarrow_r \mathsf{Sign}_\Sigma(\mathsf{sk}'_\Sigma, (\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{A}, m_{!\mathsf{A}}, h, \mathbb{A}))$, and $c \leftarrow_r \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_1)$. Let $\pi \leftarrow_r \mathsf{Prove}_\Omega\{(x_1, x_2, \sigma_{\mathsf{sk}_{\mathsf{S}}}, \sigma_{\mathsf{sk}_{\mathbb{S}}}) : (y_1 = f(x_1) \wedge c = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_1)) \vee (y_2 = f(x_2) \wedge c = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_2) \wedge \mathsf{Verf}_\Sigma(\mathsf{pk}_\Sigma, (y_2, \mathsf{pk}_{\mathsf{P3S}}), \sigma_{\mathsf{sk}_{\mathbb{S}}}) = 1)\}(\ell)$, where $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c)$. Return $\sigma \leftarrow (h, r, \mathsf{A}, \sigma_m, \mathbb{A}, \pi, c)$.

$\underline{\mathsf{AddSan}_{\mathsf{P3S}}(\mathsf{sk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathbb{S})}$: If $\mathbb{S} \notin 2^{\mathbb{U}}$, return $\perp$. Let $\sigma_{\mathsf{sk}_{\mathbb{S}}} \leftarrow_r \mathsf{Sign}_\Sigma(\mathsf{sk}_\Sigma, (\mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}))$ and $\mathsf{sk}'_{\mathbb{S}} \leftarrow_r \mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \mathbb{S})$. Return $\mathsf{sk}_{\mathbb{S}} \leftarrow (\sigma_{\mathsf{sk}_{\mathbb{S}}}, \mathsf{sk}'_{\mathbb{S}})$.

$\underline{\mathsf{Verify}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma, m)}$: If π or σ_m is not valid, return $\perp$. Check that $m_{!\mathsf{A}}$ is contained in m in the correct sequence at the right positions (derivable from A). If $\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, r, h) = 1$, return 1. Otherwise, return 0.

$\underline{\mathsf{Sanitize}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{sk}_{\mathbb{S}}, m, \sigma, \mathsf{M})}$: If $\sigma_{\mathsf{sk}_{\mathbb{S}}}$ or σ is not valid, return $\perp$. Let $r' \leftarrow_r \mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, \mathsf{sk}'_{\mathbb{S}}, m, \mathsf{M}(m), h, r)$, $c' \leftarrow_r \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_2)$, and $\pi' \leftarrow_r \mathsf{Prove}_\Omega\{(x_1, x_2, \sigma_{\mathsf{sk}_{\mathbb{S}}}) : (y_1 = f(x_1) \wedge c' = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_1)) \vee (y_2 = f(x_2) \wedge c' = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_2) \wedge \mathsf{Verf}_\Sigma(\mathsf{pk}_\Sigma, (y_2, \mathsf{pk}_{\mathsf{P3S}}), \sigma) = 1)\}(\ell)$, where $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, h, r', \mathsf{M}(m), \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c')$. Let $(\sigma', m') \leftarrow ((h, r', \mathsf{A}, \sigma_m, \mathbb{A}, \pi', c'), \mathsf{M}(m))$. If (σ', m') is not valid, return $\perp$. Return (σ', m').

$\underline{\mathsf{Proof}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma, m)}$: If σ is not valid, return $\perp$. Let $\mathsf{pk} \leftarrow \mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, c)$. Let $\pi_{\mathsf{P3S}} \leftarrow_r \mathsf{Prove}_\Omega\{(\mathsf{sk}_\Pi) : \mathsf{pk} = \mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, c) \wedge \mathsf{KVrf}_\Pi(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) = 1\}(\ell)$, where $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \sigma, \mathsf{pk}, m)$. Return $(\pi_{\mathsf{P3S}}, \mathsf{pk})$.

$\underline{\mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}, \pi_{\mathsf{P3S}}, \sigma, m)}$: If σ or π_{P3S} is not valid, return 0. Return 1.

[a] Note, we need a different CRS for each language L involved. However, we keep the description short, and thus do not make this explicit.

Construction 1: Our P3S

The full proof of Theorem 1 is given in Appendix C.

Proof (Sketch). Unforgeability follows from the zero-knowledge and extractability property of the used proof system, the insider collision-resistance of the used PCH, as well as the one-wayness of f, and and unforgeability of the used signature scheme. Immutability directly follows from the unforgeability of the used signature scheme. Privacy follows from the zero-knowledge property of the used proof system, the full indistinguishability of the used PCH, and somewhat surprisingly, the uniqueness of PCH. Likewise, transparency follows from the zero-knowledge property of the used proof system, and the IND-CPA security of the used encryption scheme. The same is true for pseudonymity, but we also require, again somewhat surprising, the uniqueness of the used PCH. Both signer-accountability, and sanitizer-accountability, follow from the extractability of the used proof system, and the one-wayness of f. Proof-soundness follows from key-verifiability of the used encryption scheme, and the extractability of the used proof system. Finally, traceability follows from the extractability of the used proof system.

Instantiation. The description of Construction 1 is as compact as reasonable. For a concrete instantiation, there are some aspects which can be optimized. Currently, it seems to be advisable to stick to the elliptic curve and in particular to the type-3 bilinear group setting (a setting where we assume the SXDH assumption to hold), due to the efficiency of the CP-ABE schemes in this setting (used by the PCH). Consequently, we consider the OWF f to be simply the function $f(x) = g^x$ for $x \in \mathbb{Z}_q$ and g being a generator of a group $\mathbb{G}$ of prime order q (and in particular one of the base groups of a bilinear group). Then, as an encryption scheme to encrypt images under f and that is key-verifiable, we can use ElGamal in either of the two base groups (for completeness, we show key-verifiability of ElGamal in Appendix B). Now, the signature keys $(\mathsf{sk}'_\Sigma, \mathsf{pk}'_\Sigma)$ used by signer to produce signatures can be any arbitrary eUNF-CMA-secure scheme. In contrast, the signature scheme associated to keys $(\mathsf{sk}_\Sigma, \mathsf{pk}_\Sigma)$ used by the group manager in $\mathsf{AddSan}_{\mathsf{P3S}}$ to certify the y_2 values of sanitizers need to be chosen with care: we need a signature scheme with message space being one of the base groups of the bilinear group and thus the natural choice is a structure preserving signature scheme [1]. Moreover, the SPS needs to be compatible with efficient NIZKPOK. Due to the choice of the OWF the most natural choice for Ω are simulation-sound extractable NIZKPOK obtained via Fiat-Shamir from Σ-protocols (cf. [32]). As PCH instantiation we can use a strengthened version of the PCH by Derler et al. [30]. See Appendix B.

Efficiency. Our scheme is reasonably efficient. The group-manager only needs to create a key-pair for a PCH, while the sanitizer only needs to evaluate a one-way functions (the signer additionally needs to draw a key-pair for an encryption scheme Π). For signing, the signer needs to generate a hash, a signature, an encryption, and a simple NIZKPOK. For sanitizing, the sanitizer has to create an encryption, adapt a hash, and attaches a simple NIZKPOK. Granting sanitizing rights boils down to creating a signature and creating a key for the PCH. Verification is also straightforward: A verifier checks a signature and the NIZPOK. Likewise, proof-generation is a simple decryption and a NIZKPOK proving that decryption was done honestly. Checking a proof is verifying that proof and a signature. Thus, ignoring the NIZKPOK and the encryptions, our scheme is therefore comparable to existing, way less expressive, constructions.

5 Conclusion

In this work we have introduced the notion of policy-based sanitizable signatures, which are an extension to standard sanitizable signature schemes, along with a provably secure construction, which features, for the first time, full accountability. In our new primitive, a sanitizer is no longer appointed by the signer at signature generation, but rather can sanitize based on a set attributes it has. We leave it as open work how to add invisibility and unlinkability to our contribution.

References

1. M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. In *CRYPTO*, pages 209–236, 2010.
2. J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, a. shelat, and B. Waters. Computing on authenticated data. *J. Cryptology*, 28(2):351–395, 2015.
3. G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS*, pages 159–177, 2005.
4. G. Ateniese and B. de Medeiros. On the key exposure problem in chameleon hashes. In *SCN*, pages 165–179, 2004.
5. G. Ateniese, B. Magri, D. Venturi, and E. R. Andrade. Redactable blockchain - or - rewriting history in bitcoin and friends. In *EuroS&P*, pages 111–126, 2017.
6. M. T. Beck, J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Practical strongly invisible and strongly accountable sanitizable signatures. In *ACISP, Part I*, pages 437–452, 2017.
7. 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.
8. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73, 1993.
9. M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, pages 136–153, 2005.
10. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *S&P*, pages 321–334, 2007.
11. A. Bilzhause, H. C. Pöhls, and K. Samelin. Position paper: The past, present, and future of sanitizable and redactable signatures. In *Ares*, pages 87:1–87:9, 2017.
12. D. Boneh, D. M. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In *PKC*, pages 68–87, 2009.
13. 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.
14. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Santizable signatures: How to partially delegate control for authenticated data. In *BIOSIG*, pages 117–128, 2009.
15. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of sanitizable signatures. In *PKC*, pages 444–461, 2010.
16. C. Brzuska, H. C. Pöhls, and K. Samelin. Non-interactive public accountability for sanitizable signatures. In *EuroPKI*, pages 178–193, 2012.
17. C. Brzuska, H. C. Pöhls, and K. Samelin. Efficient and perfectly unlinkable sanitizable signatures without group signatures. In *EuroPKI*, pages 12–30, 2013.
18. X. Bultel and P. Lafourcade. Unlinkable and strongly accountable sanitizable signatures from verifiable ring signatures. In *CANS*, pages 203–226, 2017.

19. X. Bultel, P. Lafourcade, R. W. F. Lai, G. Malavolta, D. Schröder, and S. A. Thyagarajan. Efficient invisible and unlinkable sanitizable signatures. to appear at PKC 2019, 2019.
20. J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures. In *PKC, Part II*, pages 152–182, 2017.
21. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO*, pages 410–424, 1997.
22. S. Canard and A. Jambert. On extended sanitizable signature schemes. In *CT-RSA*, pages 179–194, 2010.
23. S. Canard, A. Jambert, and R. Lescuyer. Sanitizable signatures with several signers and sanitizers. In *AfricaCrypt*, pages 35–52, 2012.
24. S. Canard, F. Laguillaumie, and M. Milhau. Trapdoor sanitizable signatures and their application to content protection. In *ACNS*, pages 258–276, 2008.
25. M. Chase and A. Lysyanskaya. On signatures of knowledge. In *Crypto*, pages 78–96, 2006.
26. D. Chaum and E. van Heyst. Group signatures. In *EuroCrypt*, pages 257–265, 1991.
27. 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.
28. 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.
29. D. Derler, H. C. Pöhls, K. Samelin, and D. Slamanig. A general framework for redactable signatures and new constructions. In *ICISC*, pages 3–19, 2015.
30. D. Derler, K. Samelin, D. Slamanig, and C. Striecks. Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. In *NDSS*, 2019.
31. 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.
32. S. Faust, M. Kohlweiss, G. A. Marson, and D. Venturi. On the non-malleability of the fiat-shamir transform. In *INDOCRYPT*, pages 60–79, 2012.
33. V. Fehr and M. Fischlin. Sanitizable signcryption: Sanitization over encrypted data (full version). *ePrint*, 2015:765, 2015.
34. M. Fischlin and P. Harasser. Invisible sanitizable signatures and public-key encryption are equivalent. In *ACNS*, pages 202–220, 2018.
35. N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schröder, and M. Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In *PKC Part-I*, pages 301–330, 2016.
36. T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Crypto*, pages 10–18, 1984.
37. E. Ghosh, M. T. Goodrich, O. Ohrimenko, and R. Tamassia. Verifiable zero-knowledge order queries and updates for fully dynamic lists and trees. In *SCN*, pages 216–236, 2016.
38. J. Gong, H. Qian, and Y. Zhou. Fully-secure and practical sanitizable signatures. In *Inscrypt*, pages 300–317, 2010.
39. J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *AsiaCrypt*, pages 444–459, 2006.
40. R. Johnson, D. Molnar, D. Xiaodong Song, and D. A. Wagner. Homomorphic signature schemes. In *CT-RSA*, pages 244–262, 2002.
41. H. Krawczyk and T. Rabin. Chameleon signatures. In *NDSS*, pages 143–154, 2000.
42. S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Chameleon-hashes with dual long-term trapdoors and their applications. In *AfricaCrypt*, pages 11–32, 2018.
43. S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Protean signature schemes. In *CANS*, pages 256–276, 2018.
44. S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Fully invisible protean signatures schemes. *ePrint*, 2019:039, 2019.
45. S. Krenn, K. Samelin, and D. Sommer. Stronger security for sanitizable signatures. In *DPM/QASA*, pages 100–117, 2015.
46. J. Lai, X. Ding, and Y. Wu. Accountable trapdoor sanitizable signatures. In *ISPEC*, pages 117–131, 2013.
47. A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EuroCrypt*, pages 62–91, 2010.
48. Y. Sakai, J. C. N. Schuldt, K. Emura, G. Hanaoka, and K. Ohta. On the security of dynamic group signatures: Preventing signature hijacking. In *PKC*, pages 715–732, 2012.
49. R. Steinfeld, L. Bull, and Y. Zheng. Content extraction signatures. In *ICISC*, pages 285–304, 2001.
50. S. Yamada, N. Attrapadung, G. Hanaoka, and N. Kunihiro. Generic constructions for chosen-ciphertext secure attribute based encryption. In *PKC*, pages 71–89, 2011.
51. D. H. Yum, J. W. Seo, and P. Joong Lee. Trapdoor sanitizable signatures made easy. In *ACNS*, pages 53–68, 2010.

A More Preliminaries and Building Blocks

This section is devoted to give additional background on the building blocks.

A.1 Additional Preliminaries

We first give some additional preliminaries required to understand the concrete constructions given in Appendix B.

Unknown-Order Group Definitions and Assumptions.

RSA Key-Generator. Let $(N, p, q, e, d) \leftarrow_r \mathsf{RSAGen}(1^\kappa)$ 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 RSAGen always outputs moduli with the same bit-length, based on κ.

The One-More-RSA Inversion Assumption [7]. Let $(n, e, d, p, q) \leftarrow_r \mathsf{RSAGen}(1^\kappa)$ be an RSA-key generator returning an RSA modulus $n = pq$, where p and q are random distinct primes, $e > 1$ an integer co-prime to $\varphi(n)$, and $d \equiv e^{-1} \bmod \varphi(n)$. The one-more-RSA-assumption associated to RSAGen is provided an inversion oracle $\mathcal{I}$, which inverts any element $x \in \mathbb{Z}_n^*$ w.r.t. e, and a challenge oracle $\mathcal{C}$, which at each call returns a random element $y_i \in \mathbb{Z}_n^*$.

Definition 25 (One-More-RSA Inversion Assumption). *An adversary wins if, given n and e, it is able to invert more elements received by $\mathcal{C}$ than is makes calls to $\mathcal{I}$. The corresponding assumption states that for every PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

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

Here, X is the set of inverted challenges.

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

A.2 Additional Building Blocks

We now present our additional building blocks.

Standard Chameleon-Hashes. Chameleon-hashes behave similar to standard collision-resistant hash-functions, but allow to find arbitrary collisions, if a trapdoor is known [41].

The following framework is derived from Camenisch et al. [20].

Definition 26 (Chameleon-Hashes). *A chameleon-hash* CH *consists of five algorithms* $(\mathsf{PPGen_{CH}}, \mathsf{KGen_{CH}}, \mathsf{Hash_{CH}}, \mathsf{Verify_{CH}}, \mathsf{Adapt_{CH}})$*, such that:*

$\mathsf{PPGen_{CH}}$. *The algorithm* $\mathsf{PPGen_{CH}}$ *on input security parameter κ outputs public parameters* $\mathsf{pp_{CH}}$ *of the scheme. For brevity, we assume that* $\mathsf{pp_{CH}}$ *is implicit input to all other algorithms:*

$$\mathsf{pp_{CH}} \leftarrow_r \mathsf{PPGen_{CH}}(1^\kappa)$$

$\mathsf{KGen_{CH}}$. *The algorithm* $\mathsf{KGen_{CH}}$*, given the public parameters* $\mathsf{pp_{CH}}$*, outputs the private* $(\mathsf{sk_{CH}})$ *and public key* $(\mathsf{pk_{CH}})$ *of the scheme*

$$(\mathsf{sk_{CH}}, \mathsf{pk_{CH}}) \leftarrow_r \mathsf{KGen_{CH}}(\mathsf{pp_{CH}})$$

$\mathsf{Hash_{CH}}$. *The algorithm* $\mathsf{Hash_{CH}}$ *gets as input the public key* $\mathsf{pk_{CH}}$*, and a message m to hash. It outputs a hash h, and some randomness r:*

$$(h, r) \leftarrow_r \mathsf{Hash_{CH}}(\mathsf{pk_{CH}}, m)$$

$\mathsf{Verify_{CH}}$. *The deterministic algorithm* $\mathsf{Verify_{CH}}$ *gets as input the public key* $\mathsf{pk_{CH}}$*, a message m, randomness r, and a hash h. It outputs a decision $d \in \{0, 1\}$ indicating whether the hash h is valid:*

$$d \leftarrow \mathsf{Verify_{CH}}(\mathsf{pk_{CH}}, m, h, r)$$

$\mathsf{Adapt_{CH}}$. *The algorithm* $\mathsf{Adapt_{CH}}$ *on input of secret key* sk*, the old message* m*, the old randomness* r*, hash* h*, and a new message* m' *outputs new randomness* r'*:*

$$r' \leftarrow_r \mathsf{Adapt_{CH}}(\mathsf{sk_{CH}}, m, m', r, h)$$

Note that we assume that the $\mathsf{Adapt_{CH}}$ algorithm always verifies if the hash it is given is valid, and outputs $\perp$ otherwise.

For a CH we require the correctness property to hold. In particular, we require that for all $\kappa \in \mathbb{N}$, for all $\mathsf{pp_{CH}} \leftarrow_r \mathsf{PPGen_{CH}}(1^\kappa)$, for all $(\mathsf{sk_{CH}}, \mathsf{pk_{CH}}) \leftarrow_r \mathsf{KGen_{CH}}(\mathsf{pp_{CH}})$, for all $m \in \mathcal{M}$, for all $(h, r) \leftarrow_r \mathsf{Hash_{CH}}(\mathsf{pk}, m)$, for all $m' \in \mathcal{M}$, we have for all for all $r' \leftarrow_r \mathsf{Adapt_{CH}}(\mathsf{sk_{CH}}, m, m', r, h)$, that $1 = \mathsf{Verify_{CH}}(\mathsf{pk_{CH}}, m, h, r) = \mathsf{Verify_{CH}}(\mathsf{pk_{CH}}, m', h, r')$. This definition captures perfect correctness.

The randomness is drawn by $\mathsf{Hash_{CH}}$, and not outside. This was done to capture "private-coin" constructions [5].

Next, we present security notions of CHs.

Full Indistinguishability. Full indistinguishability requires that the randomnesses r does not reveal if it was obtained through $\mathsf{Hash_{CHET}}$ or $\mathsf{Adapt_{PCH}}$, which is captured by the $\mathsf{HashOrAdapt}$-oracle. The messages are chosen by the adversary.

We relax the indistinguishability definition by Brzuska et al. [13] to a computational version, which is enough for most use-cases, including ours. However, compared to the existing definitions in [6, 20, 30, 42], the adversary is also allowed to generate the secret keys involved.

```
$\mathbf{Exp}^{\mathsf{FIndistinguishability}}_{\mathcal{A},\mathsf{CH}}(\kappa)$
  $\mathsf{pp_{CH}} \leftarrow_r \mathsf{PPGen_{CH}}(1^\kappa)$
  $b \leftarrow_r \{0,1\}$
  $b^* \leftarrow_r \mathcal{A}^{\mathsf{HashOrAdapt}(\cdot,\cdot,\cdot,b)}(\mathsf{pp_{CH}})$
    where oracle $\mathsf{HashOrAdapt}$ on input $\mathsf{sk_{CH}}, m, m', b$:
      $(h, r) \leftarrow_r \mathsf{Hash_{CH}}(\mathsf{pk_{CH}}, m')$
      $(h', r') \leftarrow_r \mathsf{Hash_{CH}}(\mathsf{pk_{CH}}, m)$
      $r'' \leftarrow_r \mathsf{Adapt_{CH}}(\mathsf{sk_{CH}}, m, m', r', h')$
      return $\perp$, if $r'' = \perp \ \vee \ r' = \perp \ \vee r = \perp$
      if $b = 0$, return $(h, r)$
      if $b = 1$, return $(h', r'')$
  return 1, if $b^* = b$
  return 0
```

Fig. 18: CH Full Indistinguishability

We implicitly return $\perp$ in the $\mathsf{HashOrAdapt}$ oracle (in case of an error), as the adversary $\mathcal{A}$ 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 which case it sees. For similar reasons these checks are also included in other definitions.

Definition 27 **(**CH **Full Indistinguishability).** *We say a* CH *scheme is fully indistinguishable, if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* ν *such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{FIndistinguishability}}_{\mathcal{A},\mathsf{CH}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 18.

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 [4]: 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.

```
$\mathbf{Exp}^{\mathsf{Collision\text{-}Resistance}}_{\mathcal{A},\mathsf{CH}}(\kappa)$
  $\mathsf{pp}_{\mathsf{CH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$
  $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow_r \mathcal{A}^{\mathsf{Adapt}'_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{CH}})$
    where $\mathsf{Adapt}'_{\mathsf{CH}}$ on input $\mathsf{sk}_{\mathsf{CH}}, m, m', r, h$:
      $r' \leftarrow_r \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r, h)$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m, m'\}$
      return $r'$
  return 1, if $\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m^*, h^*, r^*) = \mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m'^*, h^*, r'^*) = 1 \wedge$
    $m^* \notin \mathcal{Q} \;\wedge\; m^* \neq m'^*$
  return 0
```

Fig. 19: CH Collision-resistance

Definition 28 (CH **Collision-Resistance**). *We say a* CH *scheme is collision-resistant, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{Collision\text{-}Resistance}}_{\mathcal{A},\mathsf{CH}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 19.

Uniqueness. Uniqueness requires that it be 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^*.

```
$\mathbf{Exp}^{\mathsf{Uniqueness}}_{\mathcal{A},\mathsf{CH}}(\kappa)$
  $\mathsf{pp}_{\mathsf{CH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$
  $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*) \leftarrow_r \mathcal{A}(\mathsf{pp}_{\mathsf{CH}})$
  return 1, if $\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}^*, m^*, h^*, r^*) = \mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}^*, m^*, h^*, r'^*) = 1$
    $\wedge\ r^* \neq r'^*$
  return 0
```

Fig. 20: CH Uniqueness

Definition 29 (CH **Uniqueness**). *We say a* CH *scheme is unique, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{Uniqueness}}_{\mathcal{A},\mathsf{CH}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 20.

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

Chameleon-Hashes with Ephemeral Trapdoors. We recall the notion of chameleon-hashes with ephemeral trapdoors (CHET) from [20]. This primitive is a variant of a chameleon-hash where, in addition to the long-term trapdoor, another ephemeral trapdoor etd (chosen freshly during hashing) is required to compute collisions.

Definition 30 (Chameleon-Hashes with Ephemeral Trapdoors). *A chameleon-hash with ephemeral trapdoors* CHET *is a tuple of five algorithms* $(\mathsf{PPGen}_{\mathsf{CHET}}, \mathsf{KGen}_{\mathsf{CHET}}, \mathsf{Hash}_{\mathsf{CHET}}, \mathsf{Verify}_{\mathsf{CHET}}, \mathsf{Adapt}_{\mathsf{CHET}})$*, such that:*

$\mathsf{PPGen}_{\mathsf{CHET}}$: *On input security parameter* κ*, this algorithm outputs the public parameters* $\mathsf{pp}_{\mathsf{CHET}}$.

$$\mathsf{pp}_{\mathsf{CHET}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)$$

We assume that $\mathsf{pp}_{\mathsf{CHET}}$ *implicitly defines the message space* $\mathcal{M}$.

$\mathsf{KGen}_{\mathsf{CHET}}$: *On input the public parameters* $\mathsf{pp}_{\mathsf{CHET}}$*, this algorithm outputs the long-term key pair* $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}})$*:*

$$(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CHET}}(\mathsf{pp}_{\mathsf{CHET}})$$

$\mathsf{Hash}_{\mathsf{CHET}}$: *On input the public key* $\mathsf{pk}_{\mathsf{CHET}}$ *and a message* m*, this algorithm outputs a hash* h*, corresponding randomness* r*, as well as the ephemeral trapdoor* etd*:*

$$(h, r, \mathsf{etd}) \leftarrow_r \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m)$$

$\mathsf{Verify}_{\mathsf{CHET}}$: *On input the public key* $\mathsf{pk}_{\mathsf{CHET}}$*, a message* m*, a hash* h*, and randomness* r*, this algorithm outputs a bit* $b \in \{1, 0\}$*:*

$$b \leftarrow \mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, h, r)$$

$\mathsf{Adapt}_{\mathsf{CHET}}$: *On input secret key* $\mathsf{sk}_{\mathsf{CHET}}$*, ephemeral trapdoor* etd*, a message* m*, a message* m'*, hash* h*, randomness* r*, and trapdoor information* etd*, this algorithm outputs randomness* r'*:*

$$r' \leftarrow_r \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{etd}, m, m', h, r)$$

Note that we assume that the $\mathsf{Adapt}_{\mathsf{CHET}}$ algorithm always verifies if the hash it is given is valid, and output $\perp$ otherwise.

For correctness, we require that for all $\kappa \in \mathbb{N}$, all $\mathsf{pp}_{\mathsf{CHET}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)$, all $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CHET}}(\mathsf{pp}_{\mathsf{CHET}})$, all $m, m' \in \mathcal{M}$, all $(h, r, \mathsf{etd}) \leftarrow_r \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m)$, all $r' \leftarrow_r \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{etd}, m, m', h, r)$, we have that $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, h, r) = \mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m', h, r') = 1$.

Full Indistinguishability. Full indistinguishability requires that it be intractable for outsiders to distinguish whether a given randomness corresponds to an output of $\mathsf{Hash}_{\mathsf{CHET}}$ or $\mathsf{Adapt}_{\mathsf{CHET}}$. This is captured within the $\mathsf{HashOrAdapt}$-oracle. Note, however, that—when compared to the definitions in [6, 20, 30]—the adversary can additionally generate all secret keys.

Definition 31 (CHET Full Indistinguishability). *We say a* CHET *scheme is fully indistinguishable, if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* ν *such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{FIndistinguishability}}_{\mathcal{A},\mathsf{CHET}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 21.

```
$\mathbf{Exp}^{\mathsf{FIndistinguishability}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$
  $\mathsf{pp}_{\mathsf{CHET}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)$
  $b \leftarrow_r \{0,1\}$
  $b^* \leftarrow_r \mathcal{A}^{\mathsf{HashOrAdapt}(\cdot,\cdot,\cdot,b)}(\mathsf{pp}_{\mathsf{CHET}})$
    where $\mathsf{HashOrAdapt}$ on input $\mathsf{sk}_{\mathsf{CHET}}, m, m', b$:
      let $(h_0, r_0, \mathsf{etd}_0) \leftarrow_r \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}, m')$
      let $(h_1, r_1, \mathsf{etd}_1) \leftarrow_r \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}, m)$
      let $r_1 \leftarrow_r \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{etd}_1, m, m', h_1, r_1)$
      return $\perp$, if $r_0 = \perp \ \vee \ r_1 = \perp$
      return $(h_b, r_b, \mathsf{etd}_b)$
  return $b = b^*$
```

Fig. 21: CHET Full Indistinguishability

```
$\mathbf{Exp}^{\mathsf{Public\ Collision\text{-}Resistance}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$
  $\mathsf{pp}_{\mathsf{CHET}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)$
  $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CHET}}(\mathsf{PPGen}_{\mathsf{CHET}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow_r \mathcal{A}^{\mathsf{Adapt}'_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}},\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{CHET}})$
    where $\mathsf{Adapt}'_{\mathsf{CHET}}$ on input $\mathsf{etd}, m, m', h, r$:
      $r' \leftarrow_r \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}, \mathsf{etd}, m, m', h, r)$
      if $r' \neq \perp$, let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m, m'\}$
      return $r'$
  return 1, if $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m^*, h^*, r^*) = 1 \ \wedge$
    $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m'^*, h^*, r'^*) = 1 \ \wedge$
    $m^* \notin \mathcal{Q} \ \wedge \ m^* \neq m'^*$
  return 0
```

Fig. 22: CHET Public Collision-Resistance

Public Collision-Resistance. Public collision-resistance grants the adversary access to an $\mathsf{Adapt}_{\mathsf{PCH}}$ oracle. It requires that it is intractable to produce collisions, other than the ones produced by the $\mathsf{Adapt}_{\mathsf{PCH}}$ oracle. Thus, the adversary gains access to a $\mathsf{Adapt}'_{\mathsf{CHET}}$-oracle, which also keeps track of the produced collisions, which we need to exclude to have a meaningful definition.

Definition 32 (CHET **Public Collision-Resistance).** *We say a* CHET *scheme is publicly collision-resistant, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{Public\ Collision\text{-}Resistance}}_{\mathcal{A},\mathsf{CHET}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 22.

Strong Private Collision-Resistance. Strong private collision-resistance requires that it is even intractable for the holder of the secret key sk to find collisions without knowledge of etd. Note, the adversary can obtain arbitrary collisions.

Definition 33 (CHET **Strong Private Collision-Resistance).** *We say a* CHET *scheme is strongly privately collision-resistant, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{Strong\ Private\ Collision\text{-}Resistance}}_{\mathcal{A},\mathsf{CHET}}(\kappa) = 1\right] \leq \nu(\kappa).$$

```
$\mathbf{Exp}^{\mathsf{Strong\ Private\ Collision\text{-}Resistance}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$
  $\mathsf{pp}_{\mathsf{CHET}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)$
  $\mathcal{Q} \leftarrow \emptyset$
  $i \leftarrow 0$
  $(\mathsf{pk}^*, m^*, r^*, m'^*, r'^*, h^*) \leftarrow_r \mathcal{A}^{\mathsf{Hash}'_{\mathsf{CHET}}(\cdot,\cdot),\mathsf{Adapt}'_{\mathsf{CHET}}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pp}_{\mathsf{CHET}})$
    where $\mathsf{Hash}'_{\mathsf{CHET}}$ on input $\mathsf{pk}$, $m$:
      $(h, r, \mathsf{etd}) \leftarrow_r \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}, m)$
      return $\perp$, if $r = \perp$
      $i \leftarrow i + 1$
      let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}, h, m, \mathsf{etd}, i)\}$
      return $(h, r)$
    and $\mathsf{Adapt}'_{\mathsf{CHET}}$ on input $\mathsf{sk}$, $h$, $r$, $m$, $m'$, $i$:
      return $\perp$, if $(\mathsf{pk}, h', m'', \mathsf{etd}, i) \notin \mathcal{Q}$ for some $h'$, $m''$, $\mathsf{etd}$, $\mathsf{pk}$
      $r' \leftarrow_r \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}, \mathsf{etd}, m, m', h, r)$
      if $r' \neq \perp$, let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{pk}, h', m, \mathsf{etd}, i), (\mathsf{pk}, h', m', \mathsf{etd}, i)\}$
      return $r'$
  return 1, if $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, h^*, r^*) = 1 \wedge$
    $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}^*, m'^*, h^*, r'^*) = 1 \ \wedge\ m^* \neq m'^* \ \wedge$
    $(\mathsf{pk}^*, h^*, m^*, \cdot, \cdot) \notin \mathcal{Q} \ \wedge\ (\mathsf{pk}^*, h^*, \cdot, \cdot, \cdot) \in \mathcal{Q}$
  return 0
```

Fig. 23: CHET Strong Private Collision-Resistance

The corresponding experiment is depicted in Figure 23.

Uniqueness. Uniqueness requires that it be 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^*.

```
$\mathbf{Exp}^{\mathsf{Uniqueness}}_{\mathcal{A},\mathsf{CHET}}(\kappa)$
  $\mathsf{pp}_{\mathsf{CHET}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)$
  $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*) \leftarrow_r \mathcal{A}(\mathsf{pp}_{\mathsf{CHET}})$
  return 1, if $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, h^*, r^*) = \mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}^*, m^*, h^*, r'^*) = 1$
    $\wedge\ r^* \neq r'^*$
  return 0
```

Fig. 24: CHET Uniqueness

Definition 34 (CHET Uniqueness). *We say a CHET scheme is unique, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

$$\Pr\left[\mathbf{Exp}^{\mathsf{Uniqueness}}_{\mathcal{A},\mathsf{CHET}}(\kappa) = 1\right] \leq \nu(\kappa).$$

The corresponding experiment is depicted in Figure 24.

Attribute-Based Encryption. Let us recall the description of a cipertext-policy attribute encryption scheme (ABE henceforth) [10].

Definition 35 (Ciphertext-Policy Attribute-Based Encryption). *A* ABE *scheme is a tuple of PPT algorithms* $(\mathsf{PPGen}_{\mathsf{ABE}}, \mathsf{KGen}_{\mathsf{ABE}}, \mathsf{Enc}_{\mathsf{ABE}}, \mathsf{Dec}_{\mathsf{ABE}})$ *such that:*

$\mathsf{PPGen}_{\mathsf{ABE}}(1^\kappa):$ *Takes as input a security parameter* κ *and outputs a master secret and public key* $(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{mpk}_{\mathsf{ABE}})$*:*

$$(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{mpk}_{\mathsf{ABE}}) \leftarrow_r \mathsf{PPGen}_{\mathsf{ABE}}(1^\kappa)$$

We assume that all subsequent algorithms will implicitly receive the master public key $\mathsf{mpk}_{\mathsf{ABE}}$ *(public parameters) as input which implicitly fixes a message space* $\mathcal{M}$*.*

$\mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S}):$ *Takes as input the master secret key* $\mathsf{msk}_{\mathsf{ABE}}$ *and a set of attributes* $\mathbb{S}$ *and outputs a secret key* ssk*:*

$$\mathsf{ssk} \leftarrow_r \mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$$

$\mathsf{Enc}_{\mathsf{ABE}}(m, \mathbb{A}):$ *Takes as input a message* $m \in \mathcal{M}$ *and an access structure* $\mathbb{A}$*. It outputs a ciphertext* c*:*

$$c \leftarrow_r \mathsf{Enc}_{\mathsf{ABE}}(m, \mathbb{A})$$

$\mathsf{Dec}_{\mathsf{ABE}}(\mathsf{ssk}, c):$ *Takes as input a secret key* ssk *and a ciphertext* c *and outputs a message* m *or* $\perp$ *in case decryption does not work:*

$$m \leftarrow \mathsf{Dec}_{\mathsf{ABE}}(\mathsf{ssk}, c)$$

Correctness of a ABE scheme requires that for all $\kappa \in \mathbb{N}$,for all access structures $\mathbb{A}$, all $(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{mpk}_{\mathsf{ABE}}) \leftarrow_r \mathsf{PPGen}_{\mathsf{ABE}}(1^\kappa)$, all $m \in \mathcal{M}$, all $\mathbb{S} \in \mathbb{A}$, all $\mathsf{ssk} \leftarrow_r \mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$ we have that $\mathsf{Dec}_{\mathsf{ABE}}(\mathsf{ssk}, \mathsf{Enc}_{\mathsf{ABE}}(m, \mathbb{A})) = m$.

Security of ABE*.* In the following, we recall adaptive $\mathsf{IND\text{-}CCA2}$ security, for ABE. It is derived from the definition given by Lewko et al. [47] and Derler et al. [30]. but altered for our used notation. Refer, e.g., to [50] for how to construct chosen-ciphertext secure ABEs from CPA-secure ones.

Definition 36 (ABE IND-CCA2-Security). *An* ABE *scheme is* $\mathsf{IND\text{-}CCA2}$*-secure, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *such that:*

$$\left|\Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},\mathsf{ABE}}(\kappa) = 1\right] - \tfrac{1}{2}\right| \leq \nu(\kappa)$$

The corresponding experiment is depicted in Figure 25.

B Concrete Instantiations of Primitives

We now present the instantiations of our building blocks.

Instantiation of Secure CHs. We recall a construction from [20] in Construction 2.

Theorem 2. *If the one-more-RSA inversion assumption [7] holds, then the construction of a* CH *given in Construction 2 is fully indistinguishable, correct, unique and collision-resistant, in the random-oracle model [8].*

Proof. All properties, but full indistinguishability, have already been proven by Camenisch et al. [20]. Thus, it remains to prove full indistinguishability.

Full Indistinguishability. We prove full indistinguishability by a sequence of games.

Game 0: The original full indistinguishability game in the case $b = 0$.
Game 1: As Game 0, but we now make the transition to $b = 1$.
Transition - Game 0 → Game 1: As there is exactly one secret key (up to the group order, which can be ignored), which makes adaption work correctly, which we explicitly check, while r is always chosen randomly, the distributions are exactly equal and thus $|\Pr[S_0] - \Pr[S_1]| = 0$ follows.

As the adversary now has to other way to win the full indistinguishability game and each hop only changes the view of the adversary negligibly, full indistinguishability is proven.

```
$\mathbf{Exp}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},\mathsf{ABE}}(\kappa)$:
  $(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{mpk}_{\mathsf{ABE}}) \leftarrow_r \mathsf{PPGen}_{\mathsf{ABE}}(1^\kappa)$
  $b \leftarrow_r \{0,1\}$
  $\mathcal{Q} \leftarrow \emptyset$
  $\mathcal{S} \leftarrow \emptyset$
  $i \leftarrow 0$
  $(m_0, m_1, \mathbb{A}^*, \mathtt{state}) \leftarrow_r \mathcal{A}^{\mathsf{KGen}'_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}},\cdot),\mathsf{KGen}''_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}},\cdot),\mathsf{Dec}'_{\mathsf{ABE}}(\cdot,\cdot)}(\mathsf{mpk}_{\mathsf{ABE}})$
    where $\mathsf{KGen}'_{\mathsf{ABE}}$ on input $\mathsf{msk}_{\mathsf{ABE}}$, $\mathbb{S}$:
      return $\mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$ and set $\mathcal{S} \leftarrow \mathcal{S} \cup \mathbb{S}$
    and $\mathsf{KGen}''_{\mathsf{ABE}}$ on input $j$, $\mathbb{S}$:
      let $\mathsf{ssk} \leftarrow_r \mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$ and set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i, \mathsf{ssk})\}$
      $i \leftarrow i + 1$
    and $\mathsf{Dec}'_{\mathsf{ABE}}$ on input $j$, $c$:
      return $\bot$, if $(j, \mathsf{ssk}) \notin \mathcal{Q}$ for some $\mathsf{ssk}$
      return $\mathsf{Dec}_{\mathsf{ABE}}(\mathsf{ssk}, c)$
  if $m_0, m_1 \notin \mathcal{M} \;\vee\; |m_0| \neq |m_1| \;\vee \mathbb{A}^* \cap \mathcal{S} \neq \emptyset$, let $c^* \leftarrow \bot$
    else let $c^* \leftarrow_r \mathsf{Enc}_{\mathsf{ABE}}(m_b, \mathbb{A}^*)$
  $b^* \leftarrow_r \mathcal{A}^{\mathsf{KGen}'''_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}},\cdot),\mathsf{KGen}''''_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}},\cdot),\mathsf{Dec}''_{\mathsf{ABE}}(\cdot,\cdot)}(c^*, \mathtt{state})$
    where $\mathsf{KGen}'''_{\mathsf{ABE}}$ on input $\mathsf{msk}_{\mathsf{ABE}}$, $\mathbb{S}$:
      return $\bot$, if $\mathbb{S} \in \mathbb{A}^*$
      return $\mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$
    and $\mathsf{KGen}''''_{\mathsf{ABE}}$ on input $j$, $\mathbb{S}$:
      let $\mathsf{ssk} \leftarrow_r \mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$ and set $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i, \mathsf{ssk})\}$
      $i \leftarrow i + 1$
    and $\mathsf{Dec}''_{\mathsf{ABE}}$ on input $j$, $c$:
      return $\bot$, if $(j, \mathsf{ssk}) \notin \mathcal{Q}$ for some $\mathsf{ssk} \;\vee\; c = c^*$
      return $\mathsf{Dec}_{\mathsf{ABE}}(\mathsf{ssk}, c)$
  if $b^* = b$ return 1 else return 0
```

Fig. 25: ABE IND-CCA2 Security

<u>$\mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$</u> : On input a security parameter κ it outputs the public parameters $\mathsf{pp}_{\mathsf{CH}} \leftarrow_r (1^\kappa, e)$, where e is prime and $e > N'$ with $N' = \max_r\{N \in \mathbb{N} : (N, \cdot, \cdot, \cdot, \cdot) \leftarrow_r \mathsf{RSAGen}(1^\kappa; r)\}$.

<u>$\mathsf{KGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$</u> : On input $\mathsf{pp}_{\mathsf{CH}} = (1^\kappa, e)$ run $(N, p, q, \cdot, \cdot) \leftarrow_r \mathsf{RSAGen}(1^\kappa)$, choose a hash-function $H : \{0,1\}^* \to \mathbb{Z}_N^*$ (modeled as a random-oracle), compute d s.t. $ed \equiv 1 \mod \varphi(N)$, set $\mathsf{sk}_{\mathsf{CH}} \leftarrow_r d$, $\mathsf{pk}_{\mathsf{CHET}} \leftarrow (N, H)$, and return $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}})$.

<u>$\mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m)$</u> : On input a public key $\mathsf{pk}_{\mathsf{CH}} = (N, H)$ and a message m, choose $r \leftarrow_r \mathbb{Z}_N^*$, compute $h \leftarrow_r H(m)r^e \mod N$ and output (h, r).

<u>$\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, h, r)$</u> : On input public key $\mathsf{pk}_{\mathsf{CH}} = (N, H)$, a message m, a hash h, and a randomness $r \in \mathbb{Z}_N^*$, it computes $h' \leftarrow_r H(m)r^e \mod N$ and outputs 1 if $h' = h$ and 0 otherwise.

<u>$\mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', h, r)$</u> : On input a secret key $\mathsf{sk}_{\mathsf{CH}} = d$, messages m and m', a hash h, and randomness values r and r', the adaptation algorithm outputs $\bot$ if $\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, h, r) \neq 1$. Otherwise, let $x \leftarrow_r H(m)$, $x' \leftarrow_r H(m')$, $y \leftarrow_r xr^e \mod N$. Output $\bot$, if $\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m', h, r') \neq 1$. Return $r' \leftarrow (y(x'^{-1}))^d \mod N$.

Construction 2: RSA-based CH

Instantiation of Secure CHETs. The generic construction is given in Construction 3. This construction is essentially the one given by Krenn et al. [42], but we additionally check whether a hash h is valid after adaption, and use the stronger CH introduced above.

Theorem 3. *If* CH *is fully indistinguishable, collision-resistant, unique, and correct, then the construction of a* CHET *given in Construction 3 is fully indistinguishable, publicly collision-resistant, strongly private collision-resistant, unique, and correct.*

$\underline{\mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)}$: On input a security parameter κ, let $\mathsf{pp}_{\mathsf{CH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$ Return $\mathsf{pp}_{\mathsf{CHET}} \leftarrow \mathsf{pp}_{\mathsf{CH}}$.
$\underline{\mathsf{KGen}_{\mathsf{CHET}}(\mathsf{pp}_{\mathsf{CHET}})}$: On input $\mathsf{pp}_{\mathsf{CHET}} = \mathsf{pp}_{\mathsf{CH}}$ run and return $(\mathsf{sk}^1_{\mathsf{CH}}, \mathsf{pk}^1_{\mathsf{CH}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$.
$\underline{\mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m)}$: On input of $\mathsf{pk}_{\mathsf{CHET}} = \mathsf{pk}^1_{\mathsf{CH}}$ and m, let $(\mathsf{etd}, \mathsf{pk}^2_{\mathsf{CH}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$. Let $(h^1, r^1) \leftarrow_r \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}^1_{\mathsf{CH}}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}))$ and $(h^2, r^2) \leftarrow_r \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}^2_{\mathsf{CH}}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}))$. Return $((h^1, h^2, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), (r^1, r^2))$.
$\underline{\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, h, r)}$: On input of $\mathsf{pk}_{\mathsf{CHET}} = \mathsf{pk}^1_{\mathsf{CH}}$, m, $h = (h^1, h^2, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}})$ and $r = (r^1, r^2)$, return 1, if $\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}^1_{\mathsf{CH}}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), h^1, r^1) = 1$ and $\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}^2_{\mathsf{CH}}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), h^2, r^2) = 1$. Otherwise, return 0.
$\underline{\mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{etd}, m, m', h, r)}$: On input a secret key $\mathsf{sk}_{\mathsf{CHET}} = \mathsf{sk}^1_{\mathsf{CH}}$, etd, messages m and m', a hash $h = (h^1, h^2, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}})$ and $r = (r^1, r^1)$, first check that $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, h, r) = 1$. Otherwise, return $\perp$. Let $r'^1 \leftarrow_r \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}^1_{\mathsf{CHET}}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), (m', \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), r^1, h^1)$ and $r'^2 \leftarrow_r \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{etd}, (m, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), (m', \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), r^2, h^2)$. Let $r' \leftarrow (r'^1, r'^2)$. If $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m', h, r') = 0$, return $\perp$. Return r'.

Construction 3: Construction of a CHET

Proof. All properties, but full indistinguishability and uniqueness, have already been proven [30, 42]. We thus prove each remaining property on its own.

Full Indistinguishability. First, we prove full indistinguishability by a sequence of games.

Game 0: The original full indistinguishability game in the case $b = 1$.
Game 1: As Game 0, but instead of calculating the hash h^1 as in the game, directly hash.
Transition - Game 0 → Game 1: We claim that Game 0 and Game 1 are indistinguishable under the full indistinguishability of CH. More formally, assume that the adversary $\mathcal{A}$ can distinguish this hop. We can then construct an adversary $\mathcal{B}$ which breaks the indistinguishability of CH. In particular, the reduction works as follows. $\mathcal{B}$ receives $\mathsf{pp}_{\mathsf{CH}}$ as it's own challenge, passing them through to $\mathcal{A}$ within $\mathsf{pp}_{\mathsf{PCH}}$ (generating the rest honestly), 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 $\mathcal{B}$. Clearly, the simulation is perfect from $\mathcal{A}$'s point of view. Note, the $\mathsf{HashOrAdapt}$ always checks if the adaption was successful, and thus so does $\mathcal{B}$, making the distributions equal. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{CH\text{-}sInd}}(\kappa)$ 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: We claim that Game 1 and Game 2 are indistinguishable under the full indistinguishability of CH. More formally, assume that the adversary $\mathcal{A}$ can distinguish this hop. We can then construct an adversary $\mathcal{B}$ which breaks the indistinguishability of CH. In particular, the reduction works as follows. $\mathcal{B}$ receives $\mathsf{pp}_{\mathsf{CH}}$ as it's own challenge, passing them through to $\mathcal{A}$ within $\mathsf{pp}_{\mathsf{PCH}}$ (generating the rest honestly), 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 $\mathcal{B}$. Clearly, the simulation is perfect from $\mathcal{A}$'s point of view. Note, the $\mathsf{HashOrAdapt}$ always checks if the adaption was successful, and thus so does $\mathcal{B}$, making the distributions equal. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{CH\text{-}sInd}}(\kappa)$ follows.
We are now in the case $b = 0$. However, as the adversary only sees negligible changes, full indistinguishability is proven.

Uniqueness. Finally, we prove uniqueness by a sequence of games.

Game 0: The original strong private collision-resistance game.
Game 1: As Game 0, but we abort if the adversary outputs $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*)$ such that the winning conditions are fulfilled. Let this event be E_1.
Transition - Game 0 → Game 1: Assume that event E_1 happens. We can then construct an adversary $\mathcal{B}$ which breaks the uniqueness of the underlying CH.
The reduction works as follows. It receives $\mathsf{pp}_{\mathsf{CH}}$ from its own challenger and embeds it into $\mathsf{pp}_{\mathsf{CHET}}$. Then, when the adversary outputs $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*)$ such that the winning conditions are fulfilled, we know that $r^*_i \neq r'^*_i$ must hold for either $i = 1$ or $i = 2$ (or even both). Thus, the adversary can return $(\mathsf{pk}'^*, (m^*, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), r^*_i, r'^*_i, h^*_i)$, where $\mathsf{pk}'^* = \mathsf{pk}^1_{\mathsf{CH}}$ if $i = 1$ and $\mathsf{pk}'^* = \mathsf{pk}^2_{\mathsf{CH}}$ otherwise, while for the hash $h^* = (h^*_1, h^*_2)$ holds. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{CH\text{-}unique}}(\kappa)$ follows.

As now the adversary has no longer the possibility to win the uniqueness game, while each hop changes the view only negligibly, uniqueness is proven.

Instantiation of Secure PCHs. Our generic construction is depicted in Construction 4. This construction is taken from [30], but we also check whether an adaption was successful.

$\underline{\mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)}$: Return $\mathsf{pp}_{\mathsf{PCH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)$.

$\underline{\mathsf{MKeyGen}_{\mathsf{PCH}}(\mathsf{pp}_{\mathsf{PCH}})}$: Return $\mathsf{sk}_{\mathsf{PCH}} \leftarrow (\mathsf{msk}_{\mathsf{ABE}}, \mathsf{sk}_{\mathsf{CHET}})$ and $\mathsf{pk}_{\mathsf{PCH}} \leftarrow (\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{pk}_{\mathsf{CHET}})$, where $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CHET}}(\mathsf{pp}_{\mathsf{PCH}})$, and $(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{mpk}_{\mathsf{ABE}}) \leftarrow_r \mathsf{PPGen}_{\mathsf{ABE}}(1^\kappa)$.

$\underline{\mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}}, \mathbb{S})}$: Parse $\mathsf{sk}_{\mathsf{PCH}}$ as $(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{sk}_{\mathsf{CHET}})$ and return $\mathsf{ssk} \leftarrow (\mathsf{sk}_{\mathsf{CHET}}, \mathsf{ssk}')$, where $\mathsf{ssk}' \leftarrow_r \mathsf{KGen}_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}}, \mathbb{S})$.

$\underline{\mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})}$: Parse $\mathsf{pk}_{\mathsf{PCH}}$ as $(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{pk}_{\mathsf{CHET}})$ and return $(h, r) \leftarrow ((h_{\mathsf{CHET}}, c), r_{\mathsf{CHET}})$, where $(h_{\mathsf{CHET}}, r_{\mathsf{CHET}}, \mathsf{etd}) \leftarrow_r \mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m)$, and $c \leftarrow_r \mathsf{Enc}_{\mathsf{ABE}}(\mathsf{etd}, \mathbb{A})$.

$\underline{\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, h, r)}$: Parse $\mathsf{pk}_{\mathsf{PCH}}$ as $(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{pk}_{\mathsf{CHET}})$, h as (h_{CHET}, c), and r as r_{CHET}. Return 1, if the following check holds and 0 otherwise $\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, (m, c), h_{\mathsf{CHET}}, r_{\mathsf{CHET}}) = 1$.

$\underline{\mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{ssk}, m, m', h, r)}$: Parse ssk as $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{ssk}')$ and h as (h_{CHET}, c), and r as r_{CHET}. Check whether $\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, h, r) = 1$ and return $\perp$ otherwise. Compute $\mathsf{etd} \leftarrow \mathsf{Dec}_{\mathsf{ABE}}(\mathsf{ssk}', c)$ and return $\perp$ if $\mathsf{etd} = \perp$. Let $r' \leftarrow r'_{\mathsf{CHET}}$, where $r'_{\mathsf{CHET}} \leftarrow_r \mathsf{Adapt}_{\mathsf{CHET}}(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{etd}, m, m', h, r_{\mathsf{CHET}})$. Return $\perp$, if $\mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m', h, r') = 0$. Return r'.

Construction 4: Black-box construction of a PCH scheme

Theorem 4. *If* ABE *is IND-CCA2-secure and correct, while* CHET *is fully indistinguishable, strongly private collision-resistant, unique, and correct, then the construction of a* PCH *given in Construction 4 is fully indistinguishable, insider collision-resistant, unique, and correct.*

Proof. Due to our strengthened notions, we need to prove each property on its own.

Uniqueness. First, we prove uniqueness by a sequence of games.

Game 0: The original uniqueness game.

Game 1: As Game 0, but we abort, if the adversary found $(\mathsf{pk}^*, m^*, r^*, r'^*, h^*)$ such that it wins the uniqueness game. Let this event be E_1.

Transition - Game 0 → Game 1: Assume towards contradiction that event E_1 happens, we can build an adversary $\mathcal{B}$ which breaks uniqueness of the underlying CHET. Our reduction receives $\mathsf{pp}_{\mathsf{CHET}}$ and embeds it into $\mathsf{pp}_{\mathsf{PCH}}$. Then, by assumption, $\mathcal{B}$ can directly return $(\mathsf{pk}_2^*, m^*, r^*, r'^*, h_0^*)$, where $\mathsf{pk}^* = (\mathsf{pk}_0^*, \mathsf{pk}_1^*)$ and $h^* = (h_0^*, c^*)$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{CHET\text{-}uniq}}(\kappa)$ follows, as c^* is part of the hash, while the randomness only applies to the CHET.

As the adversary now has no way to win the uniqueness game and the hop only changes the view of the adversary negligibly, uniqueness is proven.

Full Indistinguishability. Now, we prove full indistinguishability by a sequence of games.

Game 0: The original full indistinguishability game in the case $b = 1$.

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

Transition - Game 0 → Game 1: We claim that Game 0 and Game 1 are indistinguishable under the full indistinguishability of CHET. More formally, assume that the adversary $\mathcal{A}$ can distinguish this hop. We can then construct an adversary $\mathcal{B}$ which breaks the full indistinguishability of CHET. In particular, the reduction works as follows. $\mathcal{B}$ receives $\mathsf{pp}_{\mathsf{CHET}}$ as it's own challenge, passing them through to $\mathcal{A}$ within $\mathsf{pp}_{\mathsf{PCH}}$ (generating the rest honestly), and proceeds as in the prior game, with the exception that it uses the HashOrAdapt oracle to generate h_{CHET}. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. Clearly, the simulation is perfect from $\mathcal{A}$'s point of view. Note, the HashOrAdapt always checks if the adaption was successful, and thus so does $\mathcal{B}$, making the output behave the same. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{CHET\text{-}sInd}}(\kappa)$ follows.

We are now in the case $b = 0$. However, as the adversary only sees negligible changes, full indistinguishability is proven. Note, the ciphertext is distributed equally in all cases.

Insider Collision-Resistance. Finally, we prove insider collision-resistance by a sequence of games.

Game 0: The original strong insider collision-resistance game.

Game 1: As Game 0, but we abort, if the adversary makes a query (m, m', h, r, j), for which h verifies, to the adaption oracle, for a h returned by the hashing oracle, but m has never been input to the hashing oracle or the adaption oracle, and $\mathcal{A}$ does not have enough attributes to find a collision all by itself. Let this event be E_1.

Transition - Game 0 → Game 1: Assume that event E_1 happens with non-negligible probability. We can then construct a reduction $\mathcal{B}$ which breaks the strong private collision-resistance of the underlying CHET.

Our reduction $\mathcal{B}$ works as follows. Let q be an upper bound on the queries to the hashing oracle. The adversary $\mathcal{B}$ then makes a guess $i \leftarrow_r \{1, 2, \dots, q\}$. All queries, but the ith one, are answered as in the prior game. On the ith query, however, $\mathcal{B}$ encrypts 0 instead of the real etd. If, at some point, the adversary has asked or asks to receive ssk which would allow to decrypt that c, we abort. However, by assumption, this does not happen in at least one case, thus we at most lose a factor of q. Further assume, towards contradiction, that $\mathcal{B}$ guessed right, but $\mathcal{A}$ behaves noticeably different now. Our reduction $\mathcal{B}$ can then use $\mathcal{A}$ to break the IND-CCA2 security of the used ABE. The reduction proceeds as follows. It receives $\mathsf{mpk}_{\mathsf{ABE}}$ as its own challenge, and embeds it accordingly. The oracles are simulated as follows:

Before the challenge ciphertext is embedded on the ith query (see below), every query to $\mathsf{KGen}'_{\mathsf{PCH}}$ is answered by the $\mathsf{KGen}'_{\mathsf{ABE}}$-oracle provided. However, calls to $\mathsf{KGen}''_{\mathsf{PCH}}$ are simply stored as $(j, \mathbb{S})$ by $\mathcal{B}$. Hashing is done honestly for all queries except for the ith query, where the reduction queries its own challenger with either 0 or the correct etd, embedding the response c in the returned h. All following queries are performed honestly. After this embedding, all queries to the $\mathsf{KGen}''_{\mathsf{PCH}}$-oracle are redirected to the $\mathsf{KGen}'''_{\mathsf{ABE}}$-oracle, while queries to the $\mathsf{KGen}''_{\mathsf{PCH}}$-oracle are again stored as $(j, \mathbb{S})$. Note, by assumption $\mathcal{A}$ never queries for keys which would allow decrypting that ciphertext. Adaption is done in such a way that if h was generated by the hashing-oracle, then we only continue if $(j, \mathbb{S})$ is sufficient to decrypt (note, h is known to $\mathcal{B}$, including the access structure $\mathbb{A}$ used to generate that hash). Finally, for every decryption necessary during adaption, i.e., for ciphertexts not generated by the reduction (and ssk_j, defined by the index j, is actually sufficient to adapt; c, as part of h, never needs to be decrypted, even if it is re-used in another hash), $\mathcal{B}$ uses the provided decryption oracle to receive each etd, and proceeds like in the game. Note, adaption can still be performed honestly, as all etds are thus known. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$.

We are now in the case that etd is no longer given to the adversary $\mathcal{A}$. However, this now also means that the adversary $\mathcal{A}$ was able to find a collision, without ever having grasp on the valuable information of etd. Thus, $\mathcal{B}$ can finally use this adversary to break the strong private collision-resistance of CHET. Consider the following reduction $\mathcal{B}$: it receives $\mathsf{pp}_{\mathsf{CHET}}$ and embeds it into $\mathsf{pp}_{\mathsf{PCH}}$. $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CHET}}(\mathsf{PPGen}_{\mathsf{CHET}})$ is generated honestly. It then uses those to initialize the adversary $\mathcal{A}$. The ABE-part is done as before. The reduction $\mathcal{B}$ now proceeds as follows: every hash is generated honestly, but the ith one; here, the oracle $\mathsf{Hash}'_{\mathsf{CHET}}$ is queried. All adaptions, but the challenge one, can be performed honestly (as described above with the decryption oracle provided). For the challenge one, however, $\mathcal{B}$ uses its own oracle to find the collision. Then, if E_1 happens, $\mathcal{B}$ can return $((m^*, c^*), r^*, (m'^*, c^*), r'^*, h_0^*)$ by assumption, where $h^* = (h_0^*, c^*)$ by construction.

$|\Pr[S_0] - \Pr[S_1]| \leq q(\nu_{\mathsf{ABE\text{-}CCA2}}(\kappa) + \nu_{\mathsf{CHET\text{-}SPrivColl}}(\kappa))$ follows, where q is the number of queries to the hashing oracle.

Game 2: As Game 1, but we abort, if the adversary outputs $(m^*, r^*, m'^*, r'^*, h'^*)$, such that the winning conditions are fulfilled. Let this event be E_2.

Transition - Game 1 → Game 2: Assume that event E_2 happens with non-negligible probability. We can then construct a reduction $\mathcal{B}$ which breaks the strong private collision-resistance of the underlying CHET.

Our reduction $\mathcal{B}$ works as follows. Let q be an upper bound on the queries to hashing oracle. The adversary $\mathcal{B}$ then makes a guess $i \leftarrow_r \{1, 2, \dots, q\}$. All queries, but the ith one, are answered as in the prior game. On the ith query, however, $\mathcal{B}$ encrypts 0 instead of the real etd. If, at some point, the adversary has asked or asks to receive ssk which would allow to decrypt that c, we abort. However, by assumption, this does

not happen in at least one case, thus we at most lose a factor of q. Further assume, towards contradiction, that $\mathcal{B}$ guessed right, but $\mathcal{A}$ behaves noticeably different now. Our reduction $\mathcal{B}$ can then use $\mathcal{A}$ to break the IND-CCA2 security of the used ABE. The reduction proceeds as follows. It receives $\mathsf{mpk}_{\mathsf{ABE}}$ as its own challenge, and embeds it accordingly. The oracles are simulated as follows:
Before the challenge ciphertext is embedded on the ith query (see below), every query to $\mathsf{KGen}'_{\mathsf{PCH}}$ is answered by the $\mathsf{KGen}'_{\mathsf{ABE}}$-oracle provided. However, calls to $\mathsf{KGen}''_{\mathsf{PCH}}$ are simply stored as $(j, \mathbb{S})$ by $\mathcal{B}$. Hashing is done honestly for all queries except for the ith query, where the reduction queries its own challenger with either 0 or the correct etd, embedding the response c in the returned h. All following queries are performed honestly. After this embedding, all queries to the $\mathsf{KGen}''_{\mathsf{PCH}}$-oracle are redirected to the $\mathsf{KGen}'''_{\mathsf{ABE}}$-oracle, while queries to the $\mathsf{KGen}''_{\mathsf{PCH}}$-oracle are again stored as $(j, \mathbb{S})$. Note, by assumption, i.e., $\mathcal{A}$ never queries for key which would allow decrypting that ciphertext. Adaption is done in such a way that if h was generated by the hashing-oracle, then we only continue if $(j, \mathbb{S})$ is sufficient to decrypt (note, h is known to $\mathcal{B}$, including the access structure $\mathbb{A}$ used to generate that hash). Finally, for every decryption necessary during adaption, i.e., for ciphertexts not generated by the reduction (and ssk_j, defined by the index j, is actually sufficient to adapt; c, as part of h, never needs to be decrypted, even if it is re-used in another hash), $\mathcal{B}$ uses the provided decryption oracle to receive each etd, and proceeds like in the game. Note, adaption can still be performed honestly, as all etds are thus known. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$.
We are now in the case that etd is no longer given to the adversary $\mathcal{A}$. However, this now also means that the adversary $\mathcal{A}$ was able to find a collision, without ever having grasp on the valuable information of etd. Thus, $\mathcal{B}$ can finally use this adversary to break the strong private collision-resistance of CHET. Consider the following reduction $\mathcal{B}$: it receives $\mathsf{pp}_{\mathsf{CHET}}$ and embeds it into $\mathsf{pp}_{\mathsf{PCH}}$. $(\mathsf{sk}_{\mathsf{CHET}}, \mathsf{pk}_{\mathsf{CHET}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CHET}}(\mathsf{PPGen}_{\mathsf{CHET}})$ is generated honestly. It then uses those to initialize the adversary $\mathcal{A}$. The ABE-part is done as before. The reduction $\mathcal{B}$ now proceeds as follows: every hash is generated honestly, but the ith one; here, the oracle $\mathsf{Hash}'_{\mathsf{CHET}}$ is queried. All adaptions, but the challenge one, can be performed honestly (as described above with the decryption oracle provided). For the challenge one, however, $\mathcal{B}$ uses its own oracle to find the collision. Then, if E_2 happens, $\mathcal{B}$ can return $((m^*, c^*), r^*, (m'^*, c^*), r'^*, h_0^*)$ by assumption, where $h^* = (h_0^*, c^*)$ by construction.
$|\Pr[S_1] - \Pr[S_2]| \leq q(\nu_{\mathsf{ABE\text{-}CCA2}}(\kappa) + \nu_{\mathsf{CHET\text{-}SPrivColl}}(\kappa))$ follows, where q is the number of queries to the hashing oracle.
As now the adversary $\mathcal{A}$ has no additional way to win this game, our statement is proven.

Instantiation of a Key-Verifiable *Π*. We recall a construction from ElGamal [36] in Construction 5.

$\underline{\mathsf{PPGen}_\Pi(1^\kappa)}$: On input a security parameter κ, it outputs the public parameters $\mathsf{pp}_\Pi \leftarrow_r \mathsf{DLGen}(1^\kappa)$.
$\underline{\mathsf{KGen}_\Pi(\mathsf{pp}_\Pi)}$: On input $\mathsf{pp}_\Pi = (\mathbb{G}, g, q)$, draw $x \leftarrow_r Z_q$. Set $\mathsf{pk}_\Pi \leftarrow g^x$ and $\mathsf{sk}_\Pi \leftarrow x$. Return $(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi)$.
$\underline{\mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, m)}$: On input a public key $\mathsf{pk}_{\mathsf{CH}} = g^x$ and a message m, draw $r \leftarrow_r \mathbb{Z}_q$, compute $c_1 \leftarrow g^r$ and $c_2 \leftarrow m \cdot \mathsf{pk}_\Pi^r$. Return (c_1, c_2).
$\underline{\mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, c)}$: On input secret key $\mathsf{sk}_\Pi = x$, and a ciphertext $c = (c_1, c_2)$, compute $s \leftarrow c_1^x$ and output $m' \leftarrow c_2 s^{-1}$.
$\underline{\mathsf{KVrf}_\Pi(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi)}$: On input a secret key $\mathsf{sk}_\Pi = x$ and a public key $\mathsf{pk}_\Pi = g^{x'}$, output 1, if $g^x = \mathsf{pk}_\Pi$, and 0 otherwise.

Construction 5: ElGamal Π

Theorem 5. *If the DDH-Assumption holds in $\mathbb{G}$, then the above construction is correct, IND-CPA-secure, and key-verifiable.*

Proof. Correctness and IND-CPA-security have already been proven by ElGamal [36].
Thus, it remains to prove key-verifiability.

Key-Verifiability. We prove key-verifiability by a sequence of games.

Game 0: The original key-verifiability game.
Game 1: As Game 0, but abort, if the adversary $\mathcal{A}$ wins the game as defined. Let this event be E_1.
Transition - Game 0 → Game 1: Assume, towards contradiction, that E_1 happens. However, as we are working in prime-order groups, it is obvious that pk and sk form a one-to-one mapping ("bijective"), while the encryption scheme is perfectly correct. Thus, there is no way for the adversary to cheat here. $|\Pr[S_0]-\Pr[S_1]| = 0$ follows.
As the adversary has no more possibilities to win the game, key-verifiability is proven.

C Proof of Theorem 1

Proof. We prove each property on its own, while correctness follows from inspection.

Unforgeability. To prove insider-unforgeability, we use a sequence of games:

Game 0: The original unforgeability game.
Game 1: As Game 0, but we replace crs_Ω with the one generated by $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa, L)$ and keep the trapdoor τ.
Transition - Game 0 → Game 1: Assume towards contradiction that the adversary behaves differently. We can then build an adversary $\mathcal{B}$ which breaks the zero-knowledge property of the underlying proof-system. The reduction works as follows. Our adversary $\mathcal{B}$ receives crs_Ω from its own challenger and embeds it into $\mathsf{pp}_{\mathsf{P3S}}$ and generates all other values honestly. All proofs are then generated using the oracle P provided and embedded honestly. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{nizk\text{-}zk}}(\kappa)$ follows. Note, this also means that all proofs are now simulated, even though they still prove valid statements.
Game 2: As Game 1, but we replace crs_Ω with the one generated by $(\mathsf{crs}_\Omega, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa, L)$ and keep the trapdoors τ and ξ. Let E_2 be the event that $\mathcal{A}$ can distinguish this replacement with non-negligible probability. Moreover, note that by definition crs_Ω is exactly distributed as in the prior hop.
Transition - Game 1 → Game 2: As we only keep one additional value, i.e., ξ, this is only an internal change. $|\Pr[S_1] - \Pr[S_2]| = 0$ immediately follows.
Game 3: As Game 2, but we abort, if we cannot extract the (valid) witnesses from proofs generated by the adversary $\mathcal{A}$, i.e., for all non-simulated proofs. Let this event be E_3.
Transition - Game 2 → Game 3: Assume, towards contradiction, that event E_3 happens. We can then construct an adversary $\mathcal{B}$ which breaks the simulation-sound extractability property of the proof-system. Namely, $\mathcal{B}$ receives crs_Ω and gives it to the adversary. Every proof is generated using the oracle provided, while everything else is done honestly. If, at some point, the adversary $\mathcal{A}$ outputs a non-extractable proof π^* as part of its forgery attempt (σ^*, m^*), it can trivially be extracted by $\mathcal{B}$, breaking the simulation-sound extractability property. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ follows. Note, each proof π which belongs to a signature is bound to the message and the signature.
Game 4: As Game 3, but we abort, if the adversary was able to generate a signature σ_m^* on a string never generated by the signing-oracle. Let this event be E_4.
Transition - Game 3 → Game 4: Assume, towards contradiction, that event E_4 happens. We can then construct an adversary $\mathcal{B}$ which breaks the unforgeability of the underlying signature scheme. Namely, $\mathcal{B}$ receives pk of the signature scheme. This is embedded in pk'_Σ, while all other values are generated as in Game 3. All oracles are simulated honestly, but $\mathsf{Sign}'_{\mathsf{P3S}}$. The only change is, however, that the generation of each σ_m is outsourced to the signature-generation oracle. Then, whenever E_4 happens, $\mathcal{B}$ can return $((\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, \mathsf{A}, m_{!\mathsf{A}}, h, \mathbb{A}), \sigma_m^*)$. These values can easily be compiled using $\mathcal{A}$'s output, i.e., (m^*, σ^*). Note, this already includes that the adversary cannot temper with A. $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{eUNF\text{-}CMA}}(\kappa)$ follows.
Game 5: As Game 4, but we abort, if the adversary was able to generate (m^*, σ^*) which should not have been derivable. Let this event be E_4.

Transition - Game 4 → Game 5: Assume, towards contradiction, that event E_5 happens. We can then construct an adversary $\mathcal{B}$ which breaks the strong insider collision-resistance of the used PCH. Namely, $\mathcal{B}$ receives $\mathsf{pk}_{\mathsf{PCH}}$ of the PCH. This is embedded in $\mathsf{pk}_{\mathsf{P3S}}$, while all other values are generated as in Game 4. The GetSan-oracle is simulated honestly. Calls to $\mathsf{Sign}'_{\mathsf{P3S}}$-oracle are done honestly, but the hash is generated using the $\mathsf{Hash}'_{\mathsf{PCH}}$-oracle. Calls to the $\mathsf{AddSan}'_{\mathsf{P3S}}$-oracle are simulated as follows. If a key for a simulated sanitizer (obtained by a call to the GetSan-oracle) is to be generated, it is rerouted to $\mathsf{KGen}''_{\mathsf{PCH}}$. If the adversary wants to get a key for itself, it is re-routed to the $\mathsf{KGen}'_{\mathsf{PCH}}$-oracle and the answer embedded honestly in the response. Sanitization requests are performed honestly (but simulated proofs), with the exception that adaptions for simulated sanitizers are done using the $\mathsf{Adapt}'_{\mathsf{PCH}}$-oracle. So far, the distributions are equal. Then, whenever the adversary outputs (m^*, σ^*) such that the winning-conditions are fulfilled, our reduction $\mathcal{B}$ can return $(m^*, r^*, m'^*, r'^*, h^*)$. The values can be compiled from (m^*, σ^*) and the transcript from the signing-oracle (note, we already excluded that the adversary can temper with the hash h). $|\Pr[S_4] - \Pr[S_5]| \leq \nu_{\mathsf{PBCH\text{-}SInsider\text{-}CollRes}}(\kappa)$ follows.

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

Immutability. To prove immutability, we use a sequence of games:

Game 0: The original immutability game.

Game 1: As Game 0, we abort if the adversary outputs $(\mathsf{pk}^*, \sigma^*, m^*)$ such that the winning conditions are met. Let this event be E_1.

Transition - Game 0 → Game 1: Assume, towards contradiction, that event E_1 happens. We can then build an adversary $\mathcal{B}$ which breaks the unforgeability of the used signature scheme. Namely, we know that A (which also contains the length of the message and all non-modifiable blocks along with their location), along with $\mathsf{pk}_{\mathsf{PCH}}$, is signed. As, however, by definition, the message m^* must be different from any derivable message, A w.r.t. $\mathsf{pk}_{\mathsf{PCH}}$ was never signed in this regard. Thus, $(\mathsf{pk}^*, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{A}^*, m^*_{!\mathsf{A}}, h^*, \mathbb{A}^*)$ was never signed by the signer.

Constructing a reduction $\mathcal{B}$ is now straightforward. Our reduction $\mathcal{B}$ receives the public key pk'_{Σ} (along with the public parameters) from its own challenger. This public key is embedded as pk'_{Σ}. All other values are generated honestly. If a signature σ_m is to be generated, $\mathcal{B}$ asks its own oracle to generate that signature, embedding it into the response $\mathcal{A}$ receives. At some point, $\mathcal{A}$ returns $(\mathsf{pk}^*, \sigma^*, m^*)$. The forgery can be extracted as described above. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{eUNF\text{-}CMA}}(\kappa)$ follows.

We stress that, by construction, a sanitizer always exists. Now, the adversary can no longer win the immutability game; this game is computationally indistinguishable from the original game, which concludes the proof.

Privacy. To prove privacy, we use a sequence of games:

Game 0: The original privacy game.

Game 1: As Game 0, but we replace crs_{Ω} with the one generated by $(\mathsf{crs}_{\Omega}, \tau) \leftarrow_r \mathsf{SIM}_1(1^{\kappa}, L)$ and keep the trapdoor τ.

Transition - Game 0 → Game 1: Assume towards contradiction that the adversary behaves differently. We can then build an adversary $\mathcal{B}$ which breaks the zero-knowledge property of the underlying proof-system. The reduction works as follows. Our adversary $\mathcal{B}$ receives crs_{Ω} from its own challenger and embed it into $\mathsf{pp}_{\mathsf{P3S}}$ and generates all other values honestly. All proofs are then generated using the oracle P provided and embedded honestly. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{nizk\text{-}zk}}(\kappa)$ follows. Note, this also means that all proofs are now simulated, even though they still prove valid statements.

Game 2: As Game 1, but we replace crs_{Ω} with the one generated by $(\mathsf{crs}_{\Omega}, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^{\kappa}, L)$ and keep the trapdoors τ and ξ. Let E_2 be the event that $\mathcal{A}$ can distinguish this replacement with non-negligible probability. Moreover, note that by definition crs_{Ω} is exactly distributed as in the prior hop.

Transition - Game 1 → Game 2: As we only keep one additional value, i.e., ξ, this is only an internal change. $|\Pr[S_1] - \Pr[S_2]| = 0$ immediately follows.

Game 3: As Game 2, but we abort if (σ'_0, m) and (σ'_1, m) contain different randomness $r'_0 \neq r'_1$ if generated inside $\mathsf{LoRSanit}$. Let this event be E_3.

Transition - Game 2 → Game 3: Assume, towards contradiction, that event E_3 happens. We can then construct an adversary $\mathcal{B}$ which breaks the uniqueness of PCH. In particular, it receives $\mathsf{pp}_{\mathsf{PCH}}$ and embeds it accordingly. All other values are generated as in Game 3. Then, when $\mathcal{A}$ was able to generate $r'_0 \neq r'_1$, the reduction $\mathcal{B}$ can directly return $(\mathsf{pk}^*, m, r'_0, r'_1, h^*)$, where pk^* is contained in $\mathsf{pk}_{\mathsf{P3S}}$.
$|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{PCH\text{-}uniq}}(\kappa)$ follows, while the signature does not matter, as it is already hidden behind a simulated zero-knowledge proof, making the distributions equal.

Game 4: As Game 3, but we directly generate $(\sigma, \mathsf{M}_0(m_0))$ without using sanitizing, i.e., we freshly hash with $\mathsf{M}_0(m_0)$ (if the oracle would return a signature). Note, the proofs are already simulated, but we also need to encrypt $\mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}$, as it would be done at sanitization anyway. Moreover, the adversary never sees a non-sanitized signature from that oracle, while all proofs are already simulated.

Transition - Game 3 → Game 4: If the adversary behaves noticeably different, we can build an adversary $\mathcal{B}$ which breaks the strong indistinguishability of the used PCH. The reduction works as follows. $\mathcal{B}$ receives $\mathsf{pp}_{\mathsf{PCH}}$ and embeds is honestly. All other values are generated according to Game 3. Then, for every hash generated in the $\mathsf{LoRSanit}$-oracle, the challenge oracle is queried and the answer embedded into the response. Whatever $\mathcal{A}$ then outputs, is also output by $\mathcal{B}$. $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{PCH\text{-}SInd}}(\kappa)$ immediately follows.
We stress that, by construction, a sanitizer always exists, because $\mathbb{A} \neq \emptyset$ must hold. Thus, sanitization is always possible from any generated signature, even in the case $\mathsf{A} = (\emptyset, m_\ell)$, i.e., where a sanitizer only claims accountability, but does not modify the message itself.

Now, the privacy game is independent of the bit b, proving privacy.

Transparency. To prove transparency, we use a sequence of games:

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

Game 1: As Game 0, but we replace crs_Ω with the one generated by $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa, L)$ and keep the trapdoor τ.

Transition - Game 0 → Game 1: Assume towards contradiction that the adversary behaves differently. We can then build an adversary $\mathcal{B}$ which breaks the zero-knowledge property of the underlying proof-system. The reduction works as follows. Our adversary $\mathcal{B}$ receives crs_Ω from its own challenger and embeds it into $\mathsf{pp}_{\mathsf{P3S}}$ and generates all other values honestly. All proofs are then generated using the oracle P provided and embedded honestly. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{nizk\text{-}zk}}(\kappa)$ follows. Note, this also means that all proofs are now simulated, even though they still prove valid statements.

Game 2: As Game 1, but we replace crs_Ω with the one generated by $(\mathsf{crs}_\Omega, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa, L)$ and keep the trapdoors τ and ξ. Let E_2 be the event that $\mathcal{A}$ can distinguish this replacement with non-negligible probability. Moreover, note that by definition crs_Ω is exactly distributed as in the prior hop.

Transition - Game 1 → Game 2: As we only keep one additional value, i.e., ξ, this is only an internal change. $|\Pr[S_1] - \Pr[S_2]| = 0$ immediately follows.

Game 3: As Game 2, but we abort, if we cannot extract the (valid) witnesses from proofs generated by the adversary $\mathcal{A}$, i.e., for all non-simulated proofs. Let this event be E_3.

Transition - Game 2 → Game 3: Assume, towards contradiction, that event E_3 happens. We can then construct an adversary $\mathcal{B}$ which breaks the simulation-sound extractability property of the proof-system. Namely, $\mathcal{B}$ receives crs_Ω and gives it to the adversary. Every proof is generated using the oracle provided, while everything else is done honestly. If, at some point, the adversary $\mathcal{A}$ outputs a non-extractable proof π^* as part of its forgery attempt (σ^*, m^*), it can trivially be extracted by $\mathcal{B}$, breaking the simulation-sound extractability property. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ follows. Note, each proof π which belongs to a signature is bound to the message and the signature.

Game 4: As Game 3, but we replace the contents of c (or c' resp.) with a 0, if generated by the $\mathsf{SignOrSanit}$-oracle.

Transition - Game 3 → Game 4: Assume, towards contradiction, that the adversary behaves noticeably different. We can then construct an adversary $\mathcal{B}$ which breaks the IND-CPA-security of the used encryption scheme.

Namely, we use a series of hybrids. Our reduction $\mathcal{B}$ proceeds as follows. It receives pk_{Π} and (and the corresponding parameters) from its own challenger and embeds them correctly. All other values are generated as in Game 3. For the first i ciphertexts generated, encrypt a 0. If, however, the ith ciphertext is generated, $\mathcal{B}$ asks its own challenge oracle to either encrypt 0 or the correct value. The response is embedded to $\mathcal{B}$'s response to $\mathcal{A}$. All following ciphertexts are generated honestly. Decryption queries for "all other" ciphertexts can be queried to decryption oracle provided. Thus, Game 4.0 is the same as Game 3, while in Game 4.1., however, we make the first replacement. Then, whatever $\mathcal{A}$ outputs in Game 4.i is also output by $\mathcal{B}$. $|\Pr[S_3] - \Pr[S_4]| \leq q\nu_{\mathsf{ind\text{-}cpa}}(\kappa)$ follows, where q is the number of ciphertexts generated. We stress that we do not need to "cheat" during proof-generation, as the adversary is not allowed to query such signatures to the $\mathsf{Proof}'_{\mathsf{P3S}}$-oracle.

Game 5: As Game 5, but we directly generate $(\sigma, \mathsf{M}(m))$ without using sanitizing, i.e., we always freshly hash with $\mathsf{M}(m)$ (if the oracle would return a signature). Note, the proofs are already simulated. Moreover, the adversary never sees a non-sanitized signature from that oracle, while all proofs are already simulated.

Transition - Game 4 → Game 5: Assume, towards contradiction, that the adversary behaves noticeably different. We can build an adversary $\mathcal{B}$ which breaks the strong indistinguishability of the used PCH. The reduction works as follows. $\mathcal{B}$ receives $\mathsf{pp}_{\mathsf{PCH}}$ and embeds is honestly. All other values are generated according to Game 4. Then, for every hash generated in the $\mathsf{SignOrSanit}$ oracle the challenge oracle is queried and the answer embedded into the response. Whatever $\mathcal{A}$ then outputs, is also output by $\mathcal{B}$. $|\Pr[S_4] - \Pr[S_5]| \leq \nu_{\mathsf{PCH\text{-}SInd}}(\kappa)$ immediately follows.
We stress that, by construction, a sanitizer always exists, because $\mathbb{A} \neq \emptyset$ must hold. Thus, sanitization is always possible from any generated signature, even in the case $\mathsf{A} = (\emptyset, m_\ell)$, i.e., where a sanitizer only claims accountability.

Now, we are in the case that a signature is freshly generated. Thus, transparency is proven, as each hop only changes the view negligibly.

Pseudonymity. To prove pseudonymity, we use a sequence of games:

Game 0: The original transparency game.

Game 1: As Game 0, but we replace crs_{Ω} with the one generated by $(\mathsf{crs}_{\Omega}, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa, L)$ and keep the trapdoor τ.

Transition - Game 0 → Game 1: Assume towards contradiction that the adversary behaves differently. We can then build an adversary $\mathcal{B}$ which breaks the zero-knowledge property of the underlying proof-system. The reduction works as follows. Our adversary $\mathcal{B}$ receives crs_{Ω} from its own challenger and embeds it into $\mathsf{pp}_{\mathsf{P3S}}$ and generates all other values honestly. All proofs are then generated using the oracle P provided and embedded honestly. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{nizk\text{-}zk}}(\kappa)$ follows. Note, this also means that all proofs are now simulated, even though they still prove valid statements.

Game 2: As Game 1, but we replace crs_{Ω} with the one generated by $(\mathsf{crs}_{\Omega}, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa, L)$ and keep the trapdoors τ and ξ. Let E_2 be the event that $\mathcal{A}$ can distinguish this replacement with non-negligible probability. Moreover, note that by definition crs_{Ω} is exactly distributed as in the prior hop.

Transition - Game 1 → Game 2: As we only keep one additional value, i.e., ξ, this is only an internal change. $|\Pr[S_1] - \Pr[S_2]| = 0$ immediately follows.

Game 3: As Game 2, but we abort, if we cannot extract the (valid) witnesses from proofs generated by the adversary $\mathcal{A}$, i.e., for all non-simulated proofs. Let this event be E_3.

Transition - Game 2 → Game 3: Assume, towards contradiction, that event E_3 happens. We can then construct an adversary $\mathcal{B}$ which breaks the simulation-sound extractability property of the proof-system. Namely, $\mathcal{B}$ receives crs_{Ω} and gives it to the adversary. Every proof is generated using the oracle provided, while everything else is done honestly. If, at some point, the adversary $\mathcal{A}$ outputs a non-extractable proof π^* as part of its forgery attempt (σ^*, m^*), it can trivially be extracted by $\mathcal{B}$, breaking the simulation-sound extractability property. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ follows. Note, each proof π which belongs to a signature is bound to the message and the signature.

Game 4: As Game 3, but we abort if (σ'_0, m) and (σ'_1, m) contain different randomness $r'_0 \neq r'_1$ if generated inside $\mathsf{LoRSanit}$. Let this event be E_4.

Transition - Game 3 → Game 4: Assume, towards contradiction, that event E_4 happens. We can then construct an adversary $\mathcal{B}$ which breaks the uniqueness of PCH. In particular, it receives $\mathsf{pp}_{\mathsf{PCH}}$ and embeds it accordingly. All other values are generated as in Game 3. Then, when $\mathcal{A}$ was able to generate $r'_0 \neq r'_1$, the reduction $\mathcal{B}$ can directly return $(\mathsf{pk}^*, m, r'_0, r'_1, h^*)$, where pk^* is contained in $\mathsf{pk}_{\mathsf{P3S}}$.

$|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{PCH\text{-}uniq}}(\kappa)$ follows, while the signature does not matter, as it is already hidden behind a simulated zero-knowledge proof, making the distributions equal.

Game 5: As Game 4, but we replace the contents of c' with a 0, if generated by the $\mathsf{LoRSanit}$-oracle.

Transition - Game 4 → Game 5: Assume, towards contradiction, that the adversary behaves noticeably different. We can then construct an adversary $\mathcal{B}$ which breaks the IND-CPA-security of the used encryption scheme. Namely, we use a series of hybrids. Our reduction $\mathcal{B}$ proceeds as follows. It receives pk_{Π} and (and the corresponding parameters) from its own challenger and embeds them correctly. All other values are generated as in Game 4. For the first i ciphertexts generated, encrypt a 0. If, however, the ith ciphertext is generated, $\mathcal{B}$ asks its own challenge oracle to either encrypt 0 or the correct value. The response is embedded to $\mathcal{B}$'s response to $\mathcal{A}$. All following ciphertexts are generated honestly. Thus, Game 5.0 is the same as Game 4, while in Game 5.1., however, we make the first replacement. Then, whatever $\mathcal{A}$ outputs in Game 5.i is also output by $\mathcal{B}$.

$|\Pr[S_4] - \Pr[S_5]| \leq q\nu_{\mathsf{ind\text{-}cpa}}(\kappa)$ follows, where q is the number of queries to the $\mathsf{LoRSanit}$-oracle. We stress that we do not need to "cheat" during proof-generation, as the adversary is not allowed to query such signatures to the $\mathsf{Proof}'_{\mathsf{P3S}}$-oracle.

Now, the game is independent of the bit b, proving the theorem.

Signer-Accountability. To prove signer-accountability, we use a sequence of games:

Game 0: The original signer-accountability game.

Game 1: As Game 0, but we replace crs_{Ω} with the one generated by $(\mathsf{crs}_{\Omega}, \tau) \leftarrow_r \mathsf{SIM}_1(1^{\kappa}, L)$ and keep the trapdoor τ.

Transition - Game 0 → Game 1: Assume towards contradiction that the adversary behaves differently. We can then build an adversary $\mathcal{B}$ which breaks the zero-knowledge property of the underlying proof-system. The reduction works as follows. Our adversary $\mathcal{B}$ receives crs_{Ω} from its own challenger and embeds it into $\mathsf{pp}_{\mathsf{P3S}}$ and generates all other values honestly. All proofs are then generated using the oracle P provided and embedded honestly. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{nizk\text{-}zk}}(\kappa)$ follows. Note, this also means that all proofs are now simulated, even though they still prove valid statements.

Game 2: As Game 1, but we replace crs_{Ω} with the one generated by $(\mathsf{crs}_{\Omega}, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^{\kappa}, L)$ and keep the trapdoors τ and ξ. Let E_2 be the event that $\mathcal{A}$ can distinguish this replacement with non-negligible probability. Moreover, note that by definition crs_{Ω} is exactly distributed as in the prior hop.

Transition - Game 1 → Game 2: As we only keep one additional value, i.e., ξ, this is only an internal change. $|\Pr[S_1] - \Pr[S_2]| = 0$ immediately follows.

Game 3: As Game 2, but we abort, if we cannot extract the (valid) witnesses from proofs generated by the adversary $\mathcal{A}$, i.e., for all non-simulated proofs. Let this event be E_3.

Transition - Game 2 → Game 3: Assume, towards contradiction, that event E_3 happens. We can then construct an adversary $\mathcal{B}$ which breaks the simulation-sound extractability property of the proof-system. Namely, $\mathcal{B}$ receives crs_{Ω} and gives it to the adversary. Every proof is generated using the oracle provided, while everything else is done honestly. If, at some point, the adversary $\mathcal{A}$ outputs a non-extractable proof π^* as part of its forgery attempt (σ^*, m^*), it can trivially be extracted by $\mathcal{B}$, breaking the simulation-sound extractability property. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ follows. Note, each proof π which belongs to a signature is bound to the message and the signature.

Game 4: As Game 3, but we abort, if the adversary outputs $(\mathsf{pk}^*_0, \mathsf{pk}^*_1, \sigma^*, m^*, \pi^*)$ such that the winning conditions are met. Let this event be E_4.

Transition - Game 3 → Game 4: Assume, towards contradiction, that E_4 happened. We can then construct an adversary $\mathcal{B}$ against the one-wayness of f. The reduction works as follows. It receives f and $f(x)$. It embeds both accordingly. Note, the proofs are simulated, and thus x is not needed to be known for π_{pk}. Every sanitization is done honestly, with the exception of simulated proofs. Then, as we know that the adversary wins its game, $\mathcal{B}$ can extract a pre-image x' such that $f(x') = f(x)$, and can return it to its own challenger. $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{ow}}(\kappa)$ follows.
As now the adversary has no more possibilities to win the signer-accountability game, the theorem is proven.

Sanitizer-Accountability. To prove sanitizer-accountability, we use a sequence of games:

Game 0: The original sanitizer-accountability game.
Game 1: As Game 0, but we replace crs_Ω with the one generated by $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa, L)$ and keep the trapdoor τ.
Transition - Game 0 → Game 1: Assume towards contradiction that the adversary behaves differently. We can then build an adversary $\mathcal{B}$ which breaks the zero-knowledge property of the underlying proof-system. The reduction works as follows. Our adversary $\mathcal{B}$ receives crs_Ω from its own challenger and embeds it into $\mathsf{pp}_{\mathsf{P3S}}$ and generates all other values honestly. All proofs are then generated using the oracle P provided and embedded honestly. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{nizk\text{-}zk}}(\kappa)$ follows. Note, this also means that all proofs are now simulated, even though they still prove valid statements.
Game 2: As Game 1, but we replace crs_Ω with the one generated by $(\mathsf{crs}_\Omega, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa, L)$ and keep the trapdoors τ and ξ. Let E_2 be the event that $\mathcal{A}$ can distinguish this replacement with non-negligible probability. Moreover, note that by definition crs_Ω is exactly distributed as in the prior hop.
Transition - Game 1 → Game 2: As we only keep one additional value, i.e., ξ, this is only an internal change. $|\Pr[S_1] - \Pr[S_2]| = 0$ immediately follows.
Game 3: As Game 2, but we abort, if we cannot extract the (valid) witnesses from proofs generated by the adversary $\mathcal{A}$, i.e., for all non-simulated proofs. Let this event be E_3.
Transition - Game 2 → Game 3: Assume, towards contradiction, that event E_3 happens. We can then construct an adversary $\mathcal{B}$ which breaks the simulation-sound extractability property of the proof-system. Namely, $\mathcal{B}$ receives crs_Ω and gives it to the adversary. Every proof is generated using the oracle provided, while everything else is done honestly. If, at some point, the adversary $\mathcal{A}$ outputs a non-extractable proof π^* as part of its forgery attempt (σ^*, m^*), it can trivially be extracted by $\mathcal{B}$, breaking the simulation-sound extractability property. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ follows. Note, each proof π which belongs to a signature is bound to the message and the signature.
Game 4: As Game 3, but we abort, if the adversary outputs $(\mathsf{pk}^*, \sigma^*, m^*, \pi^*)$ such that the winning conditions are met. Let this event be E_4.
Transition - Game 3 → Game 4: Assume, towards contradiction, that E_4 happened. We can then construct an adversary $\mathcal{B}$ against the one-wayness of f. The reduction works as follows. It receives f and $f(x)$. It embeds both accordingly. Every signing and proof-generation is done honestly, with the exception of simulated proofs. Then, as we know that the adversary wins its game, $\mathcal{B}$ can extract a pre-image x' such that $f(x') = f(x)$, and can return it to its own challenger. $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{ow}}(\kappa)$ follows.
As now the adversary has no more possibilities to win the sanitizer-accountability game, the theorem is proven.

Proof-Soundness. First, we prove proof-soundness by a sequence of games.

Game 0: The original proof-soundness game.
Game 1: As Game 0, but we replace crs_Ω with the one generated by $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa, L)$ and keep the trapdoor τ.
Transition - Game 0 → Game 1: Assume towards contradiction that the adversary behaves differently. We can then build an adversary $\mathcal{B}$ which breaks the zero-knowledge property of the underlying proof-system. The reduction works as follows. Our adversary $\mathcal{B}$ receives crs_Ω from its own challenger and embeds it

into $\mathsf{pp}_{\mathsf{P3S}}$ and generates all other values honestly. Note, in this case no proofs need to be simulated. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{nizk\text{-}zk}}(\kappa)$ follows.

Game 2: As Game 1, but we replace crs_Ω with the one generated by $(\mathsf{crs}_\Omega, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa, L)$ and keep the trapdoors τ and ξ. Let E_2 be the event that $\mathcal{A}$ can distinguish this replacement with non-negligible probability. Moreover, note that by definition crs_Ω is exactly distributed as in the prior hop.

Transition - Game 1 → Game 2: As we only keep one additional value, i.e., ξ, this is only an internal change. $|\Pr[S_1] - \Pr[S_2]| = 0$ immediately follows.

Game 3: As Game 2, but we abort if the adversary outputs $((\mathsf{pk}_i^*)_{0\leq i\leq 5}, \sigma^*, m^*, \pi_0^*, \pi_1^*)$ such that $\mathsf{pk}_2 \neq \mathsf{pk}_5$, the winning conditions are met, while $\mathsf{pk}_0^* = \mathsf{pk}_2^*$ and $\mathsf{pk}_3^* = \mathsf{pk}_5^*$ holds. Let this event be E_3.

Transition - Game 2 → Game 3: Assume, towards contradiction, that event E_3 happens. We can then construct an adversary $\mathcal{B}$ against the key-verifiability of the used encryption scheme. The reduction works as follows. It receives pp_Π, and once the adversary outputs $((\mathsf{pk}_i^*)_{0\leq i\leq 5}, \sigma^*, m^*, \pi_0^*, \pi_1^*)$, $\mathcal{B}$ extracts sk_0^* from π_0^* and sk_1^* from π_1^*. Then, it can return $(\mathsf{sk}_0^*, \mathsf{sk}_1^*, \mathsf{pk}_3^*)$ as its own forgery.
$|\Pr[S_2] - \Pr[S_3]| \leq 2\nu_{\mathsf{enc\text{-}key\text{-}verf}}(\kappa)$ immediately follows.

Game 4: As Game 4, but we abort if the adversary outputs $((\mathsf{pk}_i^*)_{0\leq i\leq 5}, \sigma^*, m^*, \pi_0^*, \pi_1^*)$ for which the winning conditions are met. Let this event be E_4.

Transition - Game 3 → Game 4: If this event (E_4) happens, either π_0^* or π_1^* is a bogus proof, as at least one proves a false statement. For the reduction, $\mathcal{B}$ proceeds as in the prior game (doing everything honestly, but using crs_Ω received from $\mathcal{B}$'s own challenger) and randomly selects either the first statement (concerning $(\mathsf{pk}_i^*)_{0\leq i\leq 2}$) or the second statement (concerning $(\mathsf{pk}_i^*)_{3\leq i\leq 5}$). As $\mathcal{B}$ needs to randomly guess (one may still be true), we lose a factor of 2 in the reduction. $|\Pr[S_3] - \Pr[S_4]| \leq 2\nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ follows.
As the adversary now has to other way to win the proof-soundness game and each hop only changes the view of the adversary negligibly, proof-soundness is proven.

Traceability. Next, we prove traceability by a sequence of games.

Game 0: The original traceability game.

Game 1: As Game 0, but we replace crs_Ω with the one generated by $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa, L)$ and keep the trapdoor τ.

Transition - Game 0 → Game 1: Assume towards contradiction that the adversary behaves differently. We can then build an adversary $\mathcal{B}$ which breaks the zero-knowledge property of the underlying proof-system. The reduction works as follows. Our adversary $\mathcal{B}$ receives crs_Ω from its own challenger and embeds it into $\mathsf{pp}_{\mathsf{P3S}}$ and generates all other values honestly. All proofs are then generated using the oracle P provided and embedded honestly. Then, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{nizk\text{-}zk}}(\kappa)$ follows. Note, this also means that all proofs are now simulated, even though they still prove valid statements.

Game 2: As Game 1, but we replace crs_Ω with the one generated by $(\mathsf{crs}_\Omega, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa, L)$ and keep the trapdoors τ and ξ. Let E_2 be the event that $\mathcal{A}$ can distinguish this replacement with non-negligible probability. Moreover, note that by definition crs_Ω is exactly distributed as in the prior hop.

Transition - Game 1 → Game 2: As we only keep one additional value, i.e., ξ, this is only an internal change. $|\Pr[S_1] - \Pr[S_2]| = 0$ immediately follows.

Game 3: As Game 2, but we abort if the adversary outputs a valid $(\mathsf{pk}^*, \sigma^*, m^*)$ for which we cannot (as the holder of $\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{Sig}}$) extract a pk which makes $\mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}^*, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, \mathsf{pk}, \pi_{\mathsf{P3S}}, \sigma^*, m^*)$ output 0. Let this event be E_3.

Transition - Game 2 → Game 3: If this event (E_3) happens, we have a bogus proof π contained in σ^*, as it proves a false statement. Thus, $\mathcal{B}$ proceeds as in the prior game (doing everything honestly, but using simulated proofs and the simulated crs_Ω), and can simply return the statement claimed to be proven by π and π itself. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ directly follows.

D Relations of Security Properties

We now show several relations among the security properties defined. These relations may only hold relative to the assumptions we use in our construction.

Theorem 6 (Unforgeability is independent). *There exists a* $\mathsf{P3S}$ *which offers all security properties, but unforgeability.*

Proof. A counter-example is simple: Alter $\mathsf{AddSan}_{\mathsf{P3S}}$ in such a way, that a sanitizer receives a $\mathsf{sk}_{\mathbb{S}}$ not only for the asked for attributes, but for all attributes. Clearly, all other properties, including correctness, are still preserved, but now a sanitizer can alter more signatures than it should be allowed to, as it holds a $\mathsf{sk}_{\mathbb{S}}$ for all attributes. Moreover, it still cannot blame a signer or sanitizer for the signatures it creates.

Theorem 7 (Transparency is independent). *There exists a* $\mathsf{P3S}$ *which offers all security properties, but transparency.*

Proof. This holds by altering our construction. Namely, at signing, the label to the proof system is no longer $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c)$, but $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c, 0)$. For sanitization, the label is changed to $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c, 1)$. For verification, both possibilities (last bit equal to 0 or equal to 1; Both values are distinct, and are neither derived from any secrets or message) are tested, and only returns 1, if one of the verification procedures return 1. Clearly, all other properties still hold, while an adversary can use the last bit to decide whether a sanitization was performed or not.

Theorem 8 (Privacy is independent). *There exists a* $\mathsf{P3S}$ *which offers all security properties, but privacy.*

Proof. We prove this by slightly altering our construction. First note that, in the privacy experiment, the adversary $\mathcal{A}$ is allowed to generate $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, and thus obviously knows it. We now alter our construction in the following way; At signing, the original message (if the message space is not compatible, one can use a hash-function) is also encrypted to the signer itself as c' (note, the signer already owns an encryption key-pair), and appended to the label $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c, c')$, and is also part of the signature $\sigma' = (\sigma, c')$. Verification works as expected. Sanitization remains the same, also using c' in the augmented label, but returns c' as part of the sanitized signature. Proof-generation and the judge now also take c' into account in a straightforward manner. All properties, but privacy, remain to hold, as we only add an additional value to the label ℓ of the proof-system. However, an adversary $\mathcal{A}$ can use its secret decryption key to decrypt the *original* message (or its hash), directly contradicting the privacy requirements. Transparency continues to hold, as the message is encrypted. Note, IND-CPA is sufficient, as this value is never decrypted by an honest party.

Theorem 9 (Immutability is independent). *There exists a* $\mathsf{P3S}$ *which offers all security properties, but immutability.*

Proof. We alter the construction in the following way: An honestly generated $\mathsf{pk}_{\mathsf{PCH}}$ is augmented by appending a 0. For usage outside of ℓ for the proof-system, this bit is dropped. However, if the appended bit is a 1, the verification algorithm now also accepts, if $\mathbb{A}$, m_{A}, and $m_{!\mathsf{A}}$ are not consistent, i.e., arbitrary. Thus, an adversary can sanitize a seen signature to arbitrary ones. Again, all properties, but immutability, remain to hold: An adversary $\mathcal{A}$ simply needs to generate a bogus public key (which is never generated in the honest case), and can then alter immutable blocks.

Theorem 10 (Pseudonymity is independent). *There exists a* $\mathsf{P3S}$ *which offers all security properties, but pseudonymity.*

Proof. We first want to remind the reader that, in the pseudonymity experiment, the adversary $\mathcal{A}$ is allowed to input arbitrary signatures, while in the transparency experiment the adversary never sees a signature from the signer in the case $b = 0$ from the LeftOrRight-oracle.

We use this gap to encode the sanitizer's identity such that it can only be noticed, if a sanitized and the original signatures are available. Let l be an upper bound on the bit-length of the output of the one-way function f. Let e be an additional security parameter. We alter signing as follows: At signing, the signer chooses a random integer $i \leftarrow_r \{0,1\}^{l+e}$, and attaches it to the label ℓ' for the NIZKPOK, i.e., $\ell' = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c, i)$, yet also to the signature, i.e., $\sigma' = (\sigma, i)$. Verification simply also takes the altered values into account. At sanitization, however, i is altered by setting $i' \leftarrow i + x'_2$, where x'_2 is the binary representation of x_2. The variable i' then becomes part of the used label for the new NIZKPOK and the sanitized signatures. If e is chosen large enough, while l is a constant, the distributions remain indistinguishable in the transparency experiment. Note, all attached values are independent of the messages, thus privacy still holds.

Clearly, all properties, but pseudonymity, hold. Namely, the adversary $\mathcal{A}$ simply checks whether a chosen x_2 and i (note, the adversary $\mathcal{A}$ also chooses the corresponding secret keys in the pseudonymity experiment, and thus knows the corresponding public keys) match by checking whether i' (generated by the challenger) equals $x'_2 + i$ or not.

Theorem 11 (Signer-Accountability is independent). *There exists a* $\mathsf{P3S}$ *which offers all security properties, but signer-accountability.*

Proof. The idea is similar to the proof for showing that immutability is independent. Namely, we alter our construction as follows. At key-generation for the signer, a 0 is appended to $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$. If some of the inner keys of $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$ are used, the last bit is simply dropped for the underlying algorithms. For the judge, however, if $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$ has a trailing 1, it always outputs 1, if the key to the checked is the corresponding sanitizer one, if signature verification passes (in other words, the generated proof is ignored, but only the validity is checked). Otherwise, the original algorithm is executed.

Now, if the signer generates a key with a trailing 1, if can make the sanitizer accountable for any signature it wants. All other properties are, however, still preserved, as all keys are part of the label, which still preserves proof-soundness.

Theorem 12 (Sanitizer-Accountability is independent). *There exists a* $\mathsf{P3S}$ *which offers all security properties, but sanitizer-accountability.*

Proof. The proof follows the same line as for proving the independence of signer-accountability. Namely, we alter our construction as follows. At key-generation for the sanitizer, a 0 is appended to $\mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}$. If some of the inner keys of $\mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}$ are used, the last bit is simply dropped for the underlying algorithms. For the judge, however, if $\mathsf{pk}^{\mathsf{San}}_{\mathsf{P3S}}$ has a trailing 1, it always outputs 1, if the key to the checked is the corresponding signer one, if signature verification passes (in other words, the generated proof is ignored, but only the validity is checked). Otherwise, the original algorithm is executed.

Now, if the sanitizer generates a key with a trailing 1, if can make the signer accountable for any signature it wants. All other properties are, however, still preserved, as all keys are part of the label, which still preserves proof-soundness.

Theorem 13 (Proof-Soundness is independent). *There exists a* $\mathsf{P3S}$ *which offers all security properties, but proof-soundness.*

Proof. We alter our construction as follows. At key-generation, all keys (group, signer, and sanitizer) are appended with a 0. If an algorithm uses an inner key, that bit is ignored. Judge, however, outputs also 1 (if the corresponding signature verifies), if *all* public keys have a trailing 1. This allows the adversary to easily win the proof-soundness experiment. All other properties are still preserved, as the adversary need to control all three key-pairs to win, which is not the case in the other definitions, but privacy. Privacy, however, still continues to hold, as the message is not input to the changes in our contrived scheme.

Theorem 14 (Traceability is independent). *There exists a* $\mathsf{P3S}$ *which offers all security properties, but traceability.*

Proof. This holds by altering our construction. Namely, at signing, the label to the proof system is no longer $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c)$, but $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c, 0)$. For sanitization, the label remains the same. If, however, the last bit is a 1, judge outputs 0.

Again, all properties, but traceability, are preserved, as an adversary can simply append a 1 to the label, which an honest player would never do.