This is the full version, with fixes, of a paper which appears in CT-RSA 2020,
20th Cryptographer's Track at the RSA Conference, San Francisco, CA, USA, Feb 24 - Feb 28, 2020,
Proceedings. LNCS 12006. pages 538–563. © Springer, 2020 [SS20].
The final authenticated version is available online at https://doi.org/10.1007/978-3-030-40186-3_23.

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 are a variant of signatures which 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 versatile primitive, proven by different variants and extensions, e.g., allowing multiple sanitizers or adding new sanitizers one-by-one. However, existing constructions are very restricted regarding their flexibility in specifying potential sanitizers. We propose a different and more powerful approach: Instead of using 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 of signatures. However, it turned out that certain *controlled* modifications of signed messages are beneficial in many scenarios [ABC+15, BPS17, DDH+15, GGOT16]. Over the years, different types of signature schemes supporting such modifications have been proposed, including homomorphic signatures [ABC+15, BFKW09], redactable signatures [DPSS15, JMSW02, SBZ01], and sanitizable signatures [ACdMT05, BFF+09, BFLS10]. In this paper, we focus on sanitizable signatures (3S henceforth). In a nutshell, a *standard* 3S [ACdMT05] 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 key pair. By using the secret key, the sanitizer can derive modified messages

‡ 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.

with modifiable parts (called admissible blocks) arbitrarily updated, along with 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. [BPS17].

After 3Ss were introduced by Ateniese et al. [ACdMT05], they received a lot of attention in the recent past. The first thorough security model was given by Brzuska et al. [BFF+09] (later slightly modified by Gong et al. [GQZ10]). Their work was later extended for multiple signers/sanitizers [BFLS09, CJL12], unlinkability (meaning derived signatures cannot be linked to its origin) [BFLS10, BPS13, BL17, BLL+19, FKM+16], non-interactive public-accountability (every party can determine which party is accountable for a given valid message/signature pair) [BPS12], limiting the sanitizer to signer-chosen values [CJ10, DS15], invisibility (meaning that an outsider cannot determine which blocks of a message are sanitizable) [BCD+17, BLL+19, CDK+17, FH18], the case of strongly unforgeable signatures [KSS15], and generalizations such as merging the functionality from sanitizable and redactable signatures [KPSS18b, KPSS19]. All these extensions make 3Ss suitable for an even broader field of use-cases of (cf. [BPS17] 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 [CLM08, LDW13, YSL10]. 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 obtaining the trapdoor from the original signer before sanitizing [LDW13]. This drastically restricts the applicability of 3Ss, their flexibility, and may lead to severe problems when the specified sanitizer is not available.

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

$P = (\texttt{IT department} \wedge \texttt{admin}) \vee (\texttt{team leader})$. Now, let's say that the head of IT department 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 PCs 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 (the "group manager") can control via P who is trusted to sanitize the document if required, and there must be means to determine who performed 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 [DSSS19]. In particular, [DSSS19] discusses an application to updating/rewriting transactions (or more generally speaking objects) in blockchains by selectively replacing the hash function used to aggregate transactions (e.g., within a Merkle-tree) by a novel chameleon hash. This adds flexibility to the initial proposal of a redactable blockchain (where entire blocks can be rewritten) due to Ateniese et al. in [AMVA17]. In [DSSS19], 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 allows 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 can thus achieve stronger guarantees than in [DSSS19]. In addition to transparency, meaning that no outsider sees whether updates happened (as also achieved in [DSSS19]), using P3S provides accountability, i.e., it can be determined who conducted the update.

Contribution and Our Techniques. We introduce the notion of policy-based sanitizable signatures (P3S). The main idea is the following: At signing, the signer assigns some access-policy P with each generated signature. A sanitizer can sanitize such signatures, if, and only if, that sanitizer has a secret key satisfying the associated policy P. Sanitizers can obtain new secret keys for some attributes in a dynamic fashion by a special entity named the "group manager", essentially playing the same role as the "*issuer*" in dynamic group signatures [BSZ05].[1] The reason for this design choice stems from practical considerations: Generated sanitizing keys must only be valid for a single group; In our example mentioned above, the sanitization rights must not work for signatures for another company. However, we also allow that signers and sanitizers can re-use their keys across different groups, e.g., in an enterprise every employee can hold a single key-pair and can participate in multiple groups without generating fresh keys for every group. In our running example, this also means that, e.g., a supplier for our company could sanitize certain signatures using its long-term key (if it received the corresponding secret keys).

We provide a natural formal framework for such P3S by extending the one for 3S. We note 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. We also consider a notion analogous to opening-soundness [SSE+12]. Moreover, we propose very strict privacy notions, where even (most of) the keys are generated by the adversary, further strengthening already existing definitions [dMPPS14, FF15, KSS15].

Finally, we provide a construction of P3S which we rigorously analyze in the proposed framework. Technically, the heart of our construction is a recent primitive called policy-based chameleon hash (PCH) [DSSS19], 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 are 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 [DSSS19] combines chameleon-hashes with ephemeral trapdoors (CHET) [CDK+17] and CCA2-secure ciphertext-policy attribute-based encryption (CP-ABE) scheme. In contrast to the original PCH definition in [DSSS19], however, we have to make some minor, yet important, alterations and show that a modified construction from [DSSS19] satisfies our stronger notions. In this regard, we also strengthen the CH and CHET definitions by Camenisch et al. [CDK+17] to also cover keys generated by the adversary. We believe that this strengthened definitions are also useful in many other scenarios.

[1] If wanted, a signer can also be a group manager *simultaneously, without* sacrificing accountability.

The concrete PCH construction then requires some additional tools and tricks; In order to achieve accountability, we use an "OR-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.[2] For a detailed intuition on the construction, see Sect. 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 $m = (m^1, m^2, \ldots, m^l)$, we call $m^i \in \mathcal{M}$, where $\mathcal{M} = \{0,1\}^*$, 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 a one-way function (OWF) f, an unforgeable digital signature scheme $\Sigma = \{\mathsf{PPGen}_\Sigma, \mathsf{KGen}_\Sigma, \mathsf{Sign}_\Sigma, \mathsf{Verf}_\Sigma\}$, and an IND-CCA2-secure encryption-scheme $\Pi = \{\mathsf{PPGen}_\Pi, \mathsf{KGen}_\Pi, \mathsf{Enc}_\Pi, \mathsf{Dec}_\Pi, \mathsf{KVrf}_\Pi\}$. Key-verifiability means that for a given public key, exactly one secret key can be found (e.g., Cramer-Shoup (CS) encryption [CS98] in a setting with common group parameters suffices), while KVrf_Π checks whether a given secret key sk belongs to a pk. Moreover, we require a (labeled) simulation-sound extractable non-interactive zero-knowledge proof system $\Omega = \{\mathsf{PPGen}_\Omega,$

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

$\mathsf{Prove}_\Omega, \mathsf{Verify}_\Omega\}$, and a recent primitive dubbed policy-based chameleon-hash (PCH), recently introduced by Derler et al. [DSSS19].

For the sake of readability, a somewhat informal Camenisch and Stadler notation [CS97] 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 simulation-sound extractable non-interactive zero-knowledge proof (NIZK for short) of the plaintext g_1 contained in C (which is assumed to be public), with a non-malleable attached label $\ell \in \{0,1\}^*$. 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 (all those values are assumed to be part of π as well). This is not made explicit to increase readability.

All primitives, but PCHs, are well-known; We give the full formal definitions of the standard building blocks in App. A, and only fully restate PCHs here. In a nutshell, a $\mathsf{PCH} = (\mathsf{PPGen}_{\mathsf{PCH}}, \mathsf{MKeyGen}_{\mathsf{PCH}}, \mathsf{KGen}_{\mathsf{PCH}}, \mathsf{Hash}_{\mathsf{PCH}}, \mathsf{Verify}_{\mathsf{PCH}}, \mathsf{Adapt}_{\mathsf{PCH}})$ 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.

Before we recall PCHs, we need to define what an access structure is.

Definition 1 (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 2 (Policy-Based Chameleon-Hashes [DSSS19]). *A policy-based chameleon-hash* PCH *consists of the following 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.*

$\mathsf{PPGen}_{\mathsf{PCH}}$. *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.*

$\mathsf{MKeyGen_{PCH}}$. *On input of some global parameters* $\mathsf{pp_{PCH}}$, $\mathsf{MKeyGen_{PCH}}$ *outputs the master private and public key* $(\mathsf{sk_{PCH}}, \mathsf{pk_{PCH}})$ *of the scheme:*

$$(\mathsf{sk_{PCH}}, \mathsf{pk_{PCH}}) \leftarrow_r \mathsf{MKeyGen_{PCH}}(\mathsf{pp_{PCH}})$$

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

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

$\mathsf{Hash_{PCH}}$. *On input a public key* $\mathsf{pk_{PCH}}$*, 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*:*

$$(h, r) \leftarrow_r \mathsf{Hash_{PCH}}(\mathsf{pk_{PCH}}, m, \mathbb{A})$$

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

$$b \leftarrow \mathsf{Verify_{PCH}}(\mathsf{pk_{PCH}}, m, h, r)$$

$\mathsf{Adapt_{PCH}}$. *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'*:*

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

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

Note, we have added an additional algorithm $\mathsf{PPGen_{PCH}}$ which outputs some additional global parameters, which was not part of the original description in [DSSS19], as we work in a slightly different setting. For correctness, we require that for all $\kappa \in \mathbb{N}$, for all $\mathsf{pp_{PCH}} \leftarrow_r \mathsf{PPGen_{PCH}}(1^\kappa)$, for all $(\mathsf{sk_{PCH}}, \mathsf{pk_{PCH}}) \leftarrow_r \mathsf{MKeyGen_{PCH}}(\mathsf{pp_{PCH}})$, for all $\mathbb{A} \subseteq 2^\mathbb{U}$, for all $\mathbb{S} \in \mathbb{A}$, for all $\mathsf{sk}_\mathbb{S} \leftarrow_r \mathsf{KGen_{PCH}}(\mathsf{sk_{PCH}}, \mathbb{S})$, for all $m \in \mathcal{M}$, for all $(h, r) \leftarrow_r \mathsf{Hash_{PCH}}(\mathsf{pk_{PCH}}, m, \mathbb{A})$, for all $m' \in \mathcal{M}$, for all $r' \leftarrow_r \mathsf{Adapt_{PCH}}(\mathsf{pk_{PCH}}, \mathsf{sk}_\mathbb{S}, m, m', h, r)$, we have that $1 = \mathsf{Verify_{PCH}}(\mathsf{pk_{PCH}}, m, h, r) = \mathsf{Verify_{PCH}}(\mathsf{pk_{PCH}}, m', h, r')$.

Furthermore, we require the following security properties, where our notion of indistinguishability below is stronger than the one introduced in [DSSS19]. We also restate the black-box construction from [DSSS19] (with some minor rephrasing and slightly stronger primitives) in App. C. The security proof in our stronger model is given in App. B.

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. 1: PCH Full Indistinguishability

Definition 3 (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:*

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

The corresponding experiment is depicted in Figure 1.

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 $\mathsf{Adapt}'_{\mathsf{PCH}}$).

$\mathbf{Exp}^{\mathsf{CRIns}}_{\mathcal{A},\mathsf{PCH}}(\kappa)$
$\quad \mathsf{pp}_{\mathsf{PCH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)$
$\quad (\mathsf{sk}_{\mathsf{PCH}}, \mathsf{pk}_{\mathsf{PCH}}) \leftarrow_r \mathsf{MKeyGen}_{\mathsf{PCH}}(\mathsf{pp}_{\mathsf{PCH}})$
$\quad \mathcal{S} = \mathcal{H} = \mathcal{Q} \leftarrow \emptyset$
$\quad i \leftarrow 0$
$\quad (m^*, r^*, m'^*, r'^*, h^*) \leftarrow_r \mathcal{A}^{\mathsf{KGen}'_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}},\cdot),\mathsf{KGen}''_{\mathsf{PCH}}(\mathsf{sk}_{\mathsf{PCH}},\cdot)}_{\mathsf{Hash}'_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}},\cdot,\cdot),\mathsf{Adapt}'_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{PCH}})$
$\quad\quad$ where $\mathsf{KGen}'_{\mathsf{PCH}}$ on input $\mathsf{sk}_{\mathsf{PCH}}$, $\mathbb{S}$:
$\quad\quad\quad \mathsf{sk}_{\mathbb{S}} \leftarrow_r \mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}, \mathbb{S})$
$\quad\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbb{S}\}$
$\quad\quad\quad$ return $\mathsf{sk}_{\mathbb{S}}$
$\quad\quad$ and $\mathsf{KGen}''_{\mathsf{PCH}}$ on input $\mathsf{sk}_{\mathsf{PCH}}$, $\mathbb{S}$:
$\quad\quad\quad \mathsf{sk}_{\mathbb{S}} \leftarrow_r \mathsf{KGen}_{\mathsf{PCH}}(\mathsf{sk}, \mathbb{S})$
$\quad\quad\quad \mathcal{Q} \cup \{(i, \mathsf{sk}_{\mathbb{S}})\}$
$\quad\quad\quad i \leftarrow i + 1$
$\quad\quad\quad$ return $\perp$
$\quad\quad$ and $\mathsf{Hash}'_{\mathsf{PCH}}$ on input $\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A}$:
$\quad\quad\quad (h, r) \leftarrow_r \mathsf{Hash}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, m, \mathbb{A})$
$\quad\quad\quad$ if $r \neq \perp$, $\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, \mathbb{A}, m)\}$
$\quad\quad\quad$ return (h, r)
$\quad\quad$ and $\mathsf{Adapt}'_{\mathsf{PCH}}$ on input $\mathsf{pk}_{\mathsf{PCH}}, m, m', h, r, j$:
$\quad\quad\quad$ return $\perp$, if $(j, \mathsf{sk}_{\mathbb{S}}) \notin \mathcal{Q}$ for some $\mathsf{sk}_{\mathbb{S}}$
$\quad\quad\quad r' \leftarrow_r \mathsf{Adapt}_{\mathsf{PCH}}(\mathsf{pk}_{\mathsf{PCH}}, \mathsf{sk}_{\mathbb{S}}, m, m', h, r)$
$\quad\quad\quad$ if $r' \neq \perp \ \wedge \ (h, \mathbb{A}, m) \in \mathcal{H}$ for some $\mathbb{A}$, $\mathcal{H} \leftarrow \mathcal{H} \cup \{(h, \mathbb{A}, m')\}$
$\quad\quad\quad$ return r'
$\quad$ return 1, if
$\quad\quad \mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}, m^*, h^*, r^*) = \mathsf{Verify}_{\mathsf{PCH}}(\mathsf{pk}, m'^*, h^*, r'^*) = 1 \ \wedge$
$\quad\quad (h^*, \mathbb{A}, \cdot) \in \mathcal{H}$, for some $\mathbb{A} \ \wedge \ m^* \neq m'^* \ \wedge \ \mathbb{A} \cap \mathcal{S} = \emptyset \ \wedge \ (h^*, \cdot, m^*) \notin \mathcal{H}$
$\quad$ return 0

Fig. 2: PCH Insider Collision-Resistance

Definition 4 (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 2.

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.

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

Fig. 3: PCH Uniqueness

Definition 5 (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 3.

Note, we do not require the outsider collision-resistance notion from [DSSS19].

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 [BCD+17, BFF+09, CDK+17]. 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 [BCD+17, BFF+09, CDK+17]. The main idea is following the line of reasoning of group signatures. Namely, a designated entity, which we name "the group manager" 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 signer-chosen group, identified by a public key. Moreover, signers 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 "authorized" by the manager of that group can then sanitize the generated signatures. Moreover, in contrast to group signatures, *only the signer* can decide which party has generated a signature, essentially it is also the "opener" in group signatures, but the group manager has no opening capabilities. These proofs, however, can be verified by anyone. We keep the wording of the algorithms mostly consistent with existing work to ease readability [BFF+09].

Definition 6 (P3S)**.** *A sanitizable signature with attribute-based sanitizing* 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* 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 $l \in \mathbb{N}$, for all $m \in \mathcal{M}^l$, 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 $\mathsf{pk} = \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$. We also 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 $\mathsf{sk}_{\mathbb{S}} \leftarrow_r \mathsf{AddSan}_{\mathsf{P3S}}(\mathsf{sk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{San}}, \mathbb{S})$, for all $\mathsf{M} \in \{\mathsf{M}_i \mid \mathsf{M}_i(\mathsf{A}) = 1\}$, for all $(\sigma', m') \leftarrow_r \mathsf{Sanitize}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, \mathsf{sk}_{\mathsf{P3S}}^{\mathsf{San}}, \mathsf{sk}_{\mathbb{S}}, m, \sigma, \mathsf{M})$ we have that $\mathsf{Verify}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, \sigma', m') = 1$ and that for all $(\pi'_{\mathsf{P3S}}, \mathsf{pk}') \leftarrow_r \mathsf{Proof}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}_{\mathsf{P3S}}^{\mathsf{Sig}}, \sigma', m')$, we have that $\mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{San}}, \pi'_{\mathsf{P3S}}, \sigma', m') = 1$ and $\mathsf{pk}' = \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{San}}$.

Security Definitions. We now introduce our security definitions. To increase readability, we keep the naming close to the already existing definitions for standard 3Ss [BFF+09]. 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. This also has the effect that the implications and separations by Brzuska et al. [BFF+09] have to be revisited.

Overview. We first briefly introduce each security notion to ease understanding of the formal definitions given afterwards.

- **Unforgeability.** Unforgeability requires that an adversary cannot (except with negligible probability) generate a valid signature for some message, if it does not hold enough attributes to do so. We explicitly include the case that the adversary can be group manager of other groups, but the challenge one.
- **Immutability.** Immutability requires that an adverserial group manager cannot (except with negligible probability) create signatures with altered immutable parts. This also includes appending or removing blocks.
- **Privacy.** Privacy requires that an adversary does not learn (except with negligible probability) anything about sanitized parts, even if it can generate all keys.
- **Transparency.** Transparency requires that an adversary cannot decide (except with negligible probability) whether it sees a freshly signed signature or a sanitized one, even if it can generate all keys, but the signer's one.
- **Pseudonymity.** Pseudonymity requires that an adversary does not learn (except with negligible probability) which party is accountable for a given sanitized signature, even if it can generate all keys, but the signer's one.
- **Signer-Accountability.** Signer-Accountability requires that an adversary cannot (except with negligible probability) blame an honest sanitizer for a signature it did not create, even if it can generate all keys but the sanitizer's one.

- **Sanitizer-Accountability.** Sanitzer-Accountability requires that an adversary cannot (except with negligible probability) blame an honest signer for a signature it did not create, even if it can generate all keys but the signer's one.
- **Proof-Soundness.** Proof-Soundness requires that an adversary cannot (except with negligible probability) generate a proof for an adverserially chosen signature/message pair that points to different entities, even if it can generate all keys.
- **Traceability.** Traceability requires that an adversary cannot (except with negligible probability) generate a verifying signature such that an honest signer cannot identify the accountable party, even if it can generate all keys, but the signer's one.

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 by checking whether the element could have been derived using A and $\mathbb{A}$), and the adversary $\mathcal{A}$ does not hold enough attributes itself.[3]

Definition 7 (P3S **Unforgeability).** *We say a* P3S *scheme is unforgeable, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν*

[3] Compared to the original definition in the prior versions of this paper, we have slightly changed the winning conditions. Namely, an adversary which has never queried the $\mathsf{AddSan}'_{\mathsf{P3S}}$ oracle with the challengers' key cannot generate a signature, while if it knows a sanitizer key, it can generate new signatures, but not on non-derivable messages.

$\mathbf{Exp}^{\mathsf{Unforgeability}}_{\mathcal{A},\mathsf{P3S}}(\kappa)$

$\mathsf{pp_{P3S}} \leftarrow_r \mathsf{ParGen_{P3S}}(1^\kappa)$

$(\mathsf{sk_{P3S}}, \mathsf{pk_{P3S}}) \leftarrow_r \mathsf{Setup_{P3S}}(\mathsf{pp_{P3S}})$

$(\mathsf{sk^{Sig}_{P3S}}, \mathsf{pk^{Sig}_{P3S}}) \leftarrow_r \mathsf{KGenSig_{P3S}}(\mathsf{pp_{P3S}})$

$\mathcal{Q} = \mathcal{S} = \mathcal{R} = \mathcal{M} = \mathcal{Z} \leftarrow \emptyset$

$i \leftarrow 0$

$(m^*, \sigma^*) \leftarrow_r \mathcal{A}^{\mathsf{Sign'_{P3S}}(\cdot,\mathsf{sk^{Sig}_{P3S}},\cdot,\cdot,\cdot),\mathsf{GetSan}(),\mathsf{AddSan'_{P3S}}(\mathsf{sk_{P3S}},\cdot,\cdot),\mathsf{Sanitize'_{P3S}}(\mathsf{pk_{P3S}},\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)}_{,\mathsf{Sanitize''_{P3S}}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{Proof_{P3S}}(\cdot,\mathsf{sk^{Sig}_{P3S}},\cdot,\cdot)}(\mathsf{pk_{P3S}}, \mathsf{pk^{Sig}_{P3S}})$

where $\mathsf{Sign'_{P3S}}$ on input $\mathsf{pk'_{P3S}}$, $\mathsf{sk^{Sig}_{P3S}}$, m, A, $\mathbb{A}$:

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

if $\mathsf{pk'_{P3S}} = \mathsf{pk_{P3S}} \ \wedge \ \sigma \neq \perp$:

$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\sigma, m, \mathbb{A}, \mathsf{A})\}$

if $\mathbb{A} \in \mathcal{R}$, $\mathcal{M} \leftarrow \mathcal{M} \cup \{\mathsf{M}(m) \mid \mathsf{M}(\mathsf{A}) = 1\}$

return σ

and GetSan:

$(\mathsf{sk^{San}_{P3S}}, \mathsf{pk^{San}_{P3S}}) \leftarrow_r \mathsf{KGenSan_{P3S}}(\mathsf{pp_{P3S}})$

$\mathcal{S} \leftarrow \mathcal{S} \cup \{(\mathsf{sk^{San}_{P3S}}, \mathsf{pk^{San}_{P3S}})\}$

return $\mathsf{pk^{San}_{P3S}}$

and $\mathsf{AddSan'_{P3S}}$ on input $\mathsf{sk_{P3S}}$, $\mathsf{pk^{San}_{P3S}}$, $\mathbb{S}$

if $\neg\exists(\cdot, \mathsf{pk^{San}_{P3S}}) \in \mathcal{S}$:

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

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 $\mathsf{sk}_{\mathbb{S}}$

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

$\mathcal{Z} \leftarrow \mathcal{Z} \cup \{(i, \mathsf{sk}_{\mathbb{S}})\}$

$i \leftarrow i + 1$

return $(i-1, \mathsf{pk^{San}_{P3S}})$

and $\mathsf{Sanitize'_{P3S}}$ on input $\mathsf{pk_{P3S}}$, $\mathsf{pk^{Sig}_{P3S}}$, $\mathsf{pk^{San}_{P3S}}$, j, m, σ, M:

return $\perp$, if $\neg\exists(\mathsf{sk^{San}_{P3S}}, \mathsf{pk^{San}_{P3S}}) \in \mathcal{S}$ for some $\mathsf{sk^{San}_{P3S}}$

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

where $\mathsf{sk}_{\mathbb{S}}$ is taken from $(j, \mathsf{sk}_{\mathbb{S}}) \in \mathcal{Z}$

if $\sigma' \neq \perp$:

$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\sigma', m', \perp, \perp)\}$

return σ'

and $\mathsf{Sanitize''_{P3S}}$ on input $\mathsf{pk'_{P3S}}$, $\mathsf{pk^{Sig}_{P3S}}$, $\mathsf{pk^{San}_{P3S}}$, $\mathsf{sk}_{\mathbb{S}}$, m, σ, M:

return $\perp$, if $\neg\exists(\mathsf{sk^{San}_{P3S}}, \mathsf{pk^{San}_{P3S}}) \in \mathcal{S} \ \vee \ \mathsf{pk'_{P3S}} = \mathsf{pk_{P3S}}$

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

return σ'

return 0, if $\mathsf{Verify_{P3S}}(\mathsf{pk_{P3S}}, \mathsf{pk^{Sig}_{P3S}}, \sigma^*, m^*) = 0 \ \vee \ m^* \in \mathcal{M}$

return 1, if $((\sigma^*, m^*, \cdot, \cdot) \notin \mathcal{Q} \ \wedge \ \mathcal{R} = \emptyset) \ \vee \ (\cdot, m^*, \cdot, \cdot) \notin \mathcal{Q}$

return 0

Fig. 4: P3S Unforgeability

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 4.

Immutability. The above unforgeability definition assumes that the holder of $\mathsf{sk}_{\mathsf{P3S}}$ (the group manager) 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 generates any signatures which are outside the span the honest signer has endorsed for *any* combination of attributes. This is captured by 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.

Definition 8 (P3S Immutability). *We say a* 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 5.

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 [dMPPS14, FF15].

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 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

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

Fig. 5: P3S Immutability

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.

$\mathbf{Exp}^{\mathsf{Privacy}}_{\mathcal{A},\mathsf{P3S}}(\kappa)$
$\quad \mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$
$\quad b \leftarrow_r \{0,1\}$
$\quad b^* \leftarrow_r \mathcal{A}^{\mathsf{LoRSanit}(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,b)}(\mathsf{pp}_{\mathsf{P3S}})$
$\quad\quad$ where $\mathsf{LoRSanit}$ on input of $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}$, $\mathbb{A}$, m_0, m_1, M_0, M_1, A, $\mathsf{sk}_{\mathbb{S}}$, b:
$\quad\quad\quad \sigma \leftarrow_r \mathsf{Sign}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, m_b, \mathsf{A}, \mathbb{A})$
$\quad\quad\quad$ for $b' \in \{0,1\}$, $(\sigma'_{b'}, \cdot) \leftarrow_r \mathsf{Sanitize}_{\mathsf{P3S}}(\mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S}}, \mathsf{sk}_{\mathbb{S}}, m_{b'}, \sigma, \mathsf{M}_{b'})$
$\quad\quad\quad$ return $\perp$, if $\sigma'_0 = \perp \ \vee \ \sigma'_1 = \perp \ \vee \ \mathsf{A}(m_0) = 0 \ \vee$
$\quad\quad\quad\quad \mathsf{A}(m_1) = 0 \ \vee \ \mathsf{M}_0(m_0) \neq \mathsf{M}_1(m_1)$
$\quad\quad\quad$ return σ'_b
$\quad$ return 1, if $b = b^*$
$\quad$ return 0

Fig. 6: P3S Privacy

Definition 9 ($\mathsf{P3S}$ **Privacy).** *We say a* $\mathsf{P3S}$ *scheme is private, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

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

The corresponding experiment is depicted in Figure 6.

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 $(\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{Sig}},\mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}})$, can be generated by the adversary, making our definition even stronger than existing ones [dMPPS14, FF15, KSS15]. 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.

In more detail, the challenger draws a bit $b\leftarrow_r\{0,1\}$, while the parameters $\mathsf{pp}_{\mathsf{P3S}}$ and the signer's key pair $(\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{Sig}},\mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}})$ are generated honestly. The adversary gains access to three oracles: $\mathsf{Sign}_{\mathsf{P3S}}$, $\mathsf{SignOrSanit}$, and $\mathsf{Proof}'_{\mathsf{P3S}}$. The $\mathsf{Sign}_{\mathsf{P3S}}$-oracle allows the adversary to generate new signatures; the only fixed input is $\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{Sig}}$. The $\mathsf{SignOrSanit}$-oracle is the challenge oracle. It allows the adversary $\mathcal{A}$ to input $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{San}}$, $\mathbb{A}$, m, M, A, and $\mathsf{sk}_{\mathbb{S}}$ (b and $\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{Sig}}$ 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 $\mathcal{A}$ can generate a proof using the $\mathsf{Proof}'_{\mathsf{P3S}}$-oracle with signatures generated by the $\mathsf{SignOrSanit}$-oracle, which directly returns the accountable party. Thus, the adversary can only input $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{Sig}}$, σ, m for which $(\mathsf{pk}_{\mathsf{P3S}},\sigma,m)$ 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 10 ($\mathsf{P3S}$ **Transparency).** *We say a* $\mathsf{P3S}$ *scheme is transparent, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

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

The corresponding experiment is depicted in Figure 7.

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

Fig. 7: P3S Transparency

Pseudonymity. Pseudonymity prohibits that an adversary can decide which sanitizer actually is responsible for a given signature, if it does not have access to $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$. This is related to the anonymity of group signatures [CvH91]. We formalize it in the following way. The challenger draws a bit $b \leftarrow_r \{0,1\}$, generates the public parameters $\mathsf{pp}_{\mathsf{P3S}}$ and the signer's key pair $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}})$ honestly. The adversary gains access to three oracles: $\mathsf{Sign}_{\mathsf{P3S}}$, $\mathsf{LoRSanit}$, and $\mathsf{Proof}'_{\mathsf{P3S}}$. The $\mathsf{Sign}_{\mathsf{P3S}}$-oracle allows the adversary to generate new signatures; the only fixed input is $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$. The $\mathsf{LoRSanit}$-oracle is the challenge oracle. It allows the adversary $\mathcal{A}$ to input $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S},0}$, $\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S},1}$, $\mathsf{sk}_{\mathbb{S}0}$, $\mathsf{sk}_{\mathbb{S}1}$, m, and σ (b and $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$ 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 $\mathsf{sk}^{\mathsf{San}}_{\mathsf{P3S},b}$ 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 $\mathsf{Proof}'_{\mathsf{P3S}}$-oracle with signatures generated by the $\mathsf{LoRSanit}$-oracle, which clearly contradicts pseudonymity. Thus, the adversary can only input $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, σ, m for which $(\mathsf{pk}_{\mathsf{P3S}}, \sigma, m)$ was never input/output to the $\mathsf{LoRSanit}$-oracle. The goal of the adversary is to guess the bit b.

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.

$\mathbf{Exp}^{\mathsf{Pseudonymity}}_{\mathcal{A},\mathsf{P3S}}(\kappa)$
  $\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$
  $(\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}) \leftarrow_r \mathsf{KGenSig}_{\mathsf{P3S}}(\mathsf{pp}_{\mathsf{P3S}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $b \leftarrow_r \{0,1\}$
  $b^* \leftarrow_r \mathcal{A}^{\mathsf{Sign}_{\mathsf{P3S}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot,\cdot),\mathsf{Proof}'_{\mathsf{P3S}}(\cdot,\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot),\mathsf{LoRSanit}(\cdot,\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}},\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,b)}(\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}})$
    where $\mathsf{Proof}'_{\mathsf{P3S}}$ on input of $\mathsf{pk}_{\mathsf{P3S}}$, $\mathsf{sk}^{\mathsf{Sig}}_{\mathsf{P3S}}$, σ, 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, σ, M, 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 σ'_b
  return 1, if $b = b^*$
  return 0

Fig. 8: $\mathsf{P3S}$ Pseudonymity

Definition 11 **($\mathsf{P3S}$ Pseudonymity).** *We say a $\mathsf{P3S}$ scheme is pseudonymous, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

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

The corresponding experiment is depicted in Figure 8.

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.

Definition 12 **($\mathsf{P3S}$ Signer-Accountability).** *We say a $\mathsf{P3S}$ scheme is signer-accountable, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

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

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

Fig. 9: P3S Signer-Accountability

The corresponding experiment is depicted in Figure 9.

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.

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

Fig. 10: P3S Sanitizer-Accountability

Definition 13 (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 10.

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. It is related to the property of opening-soundness introduced by Sakai et al. [SSE+12] for group signatures.[4]

$\mathbf{Exp}_{\mathcal{A},\mathsf{P3S}}^{\mathsf{Proof\text{-}Soundness}}(\kappa)$
  $\mathsf{pp}_{\mathsf{P3S}} \leftarrow_r \mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)$
  $((\mathsf{pk}_i^*)_{0 \leq i \leq 5}, \sigma^*, m_0^*, m_1^*, \pi_0^*, \pi_1^*) \leftarrow_r \mathcal{A}(\mathsf{pp}_{\mathsf{P3S}})$
  return 1, if $\mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}_0^*, \mathsf{pk}_1^*, \mathsf{pk}_2^*, \pi_0^*, \sigma^*, m_0^*) = 1 \wedge$
    $\mathsf{Judge}_{\mathsf{P3S}}(\mathsf{pk}_3^*, \mathsf{pk}_4^*, \mathsf{pk}_5^*, \pi_1^*, \sigma^*, m_1^*) = 1 \wedge$
    $(\mathsf{pk}_0^*, \mathsf{pk}_1^*, \mathsf{pk}_2^*) \neq (\mathsf{pk}_3^*, \mathsf{pk}_4^*, \mathsf{pk}_5^*)$
  return 0

Fig. 11: P3S Proof-Soundness

Definition 14 (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 11.

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.

[4] Compared to the definition in the prior versions, we now also allow the adversary to output two different messages.

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

Fig. 12: P3S Traceability

Definition 15 (P3S Traceability). *We say a 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 12.

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

4 Construction

In this section we present our P3S construction. The key ingredients are our strengthened version of a policy-based chameleon-hash PCH, a labeled simulation-sound extractable non-interactive zero-knowledge proof system Ω (NIZK for short), a one-way function f as well as a key-verifiable IND-CCA2 secure public key encryption scheme[5] Π and an eUNF-CMA-secure signature scheme Σ. 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 NIZK, 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

[5] Although key-verifiability is no property often explicitly used within IND-CCA2 encryption schemes, most encryption schemes are key-verifiable. See App. C.

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 NIZK for the relation R (using as label ℓ some additional information like the admissible changes).

$$
\begin{aligned}
&(y_1, c, y_2, \mathsf{pk}_\Pi, \mathsf{pk}_\Sigma),\ (x_1, x_2, \mathsf{sk}_\Pi, r, \sigma_{\mathsf{sk}_\mathbb{S}})) \in R \iff \\
&\qquad (y_1 = f(x_1) \ \wedge\ c = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_1; r) \ \wedge\ \mathsf{KVrf}_\Pi(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) = 1) \ \vee \\
&\qquad (y_2 = f(x_2) \ \wedge\ c = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_2; r) \ \wedge\ \mathsf{Verf}_\Sigma(\mathsf{pk}_\Sigma, y_2, \sigma_{\mathsf{sk}_\mathbb{S}}) = 1).
\end{aligned}
$$

Sanitizing amounts to computing a collision for the PCH hash, updating the respective message blocks, and again attaching a NIZK for relation R. Verification is straightforward. Relation R 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 signer's secret key) can simply decrypt c and prove correct decryption.[6]

It may be 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 [BFLS10], i.e., that sanitized signatures cannot be linked to its origin, which seems to be very hard to achieve with the current construction paradigm. However, finding such a construction may have its merits. Formally, for our construction, we can show the following:

Theorem 1. *If f is a one-way function, Π is IND-CCA2 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.).*

The full proof of Theorem 1 is given in App. D.

[6] Note, in the original version of this paper, the signer did not prove correctness of its public key. However, without this step, accountability does not hold.

$\underline{\mathsf{ParGen}_{\mathsf{P3S}}(1^\kappa)}:$ On input a security parameter κ, let $\mathsf{pp}_\Pi \leftarrow_r \mathsf{PPGen}_\Pi(1^\kappa)$, $\mathsf{crs}_\Omega \leftarrow_r \mathsf{PPGen}_\Omega(1^\kappa)$.[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 $y_1 \leftarrow f(x_1)$, $(\mathsf{sk}'_\Sigma, \mathsf{pk}'_\Sigma) \leftarrow_r \mathsf{KGen}_\Sigma(\mathsf{pp}_\Sigma)$.
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 $\pi \leftarrow_r \mathsf{Prove}_\Omega\{(x_1, x_2, \mathsf{sk}_\Pi, \sigma_{\mathsf{sk}_\mathbb{S}}) : (y_1 = f(x_1) \ \wedge\ c = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_1) \ \wedge\ \mathsf{KVrf}_\Pi(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) = 1) \ \vee\ (y_2 = f(x_2) \ \wedge\ c = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_2) \ \wedge\ \mathsf{Verf}_\Sigma(\mathsf{pk}_\Sigma, (y_2, \mathsf{pk}_{\mathsf{P3S}}), \sigma_{\mathsf{sk}_\mathbb{S}}) = 1)\}(\ell)$, 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 $\pi' \leftarrow_r \mathsf{Prove}_\Omega\{(x_1, x_2, \mathsf{sk}_\Pi, \sigma_{\mathsf{sk}_\mathbb{S}}) : (y_1 = f(x_1) \wedge c' = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_1) \wedge \mathsf{KVrf}_\Pi(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) = 1) \vee (y_2 = f(x_2) \ \wedge\ c' = \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, y_2) \ \wedge\ \mathsf{Verf}_\Sigma(\mathsf{pk}_\Sigma, (y_2, \mathsf{pk}_{\mathsf{P3S}}), \sigma_{\mathsf{sk}_\mathbb{S}}) = 1)\}(\ell)$, where $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}, h, r', \mathsf{M}(m), \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c')$. 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 $\pi_{\mathsf{P3S}} \leftarrow_r \mathsf{Prove}_\Omega\{(\mathsf{sk}_\Pi, x_1) : \mathsf{pk} = \mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, c) \wedge \mathsf{KVrf}_\Pi(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) = 1 \wedge y_1 = f(x_1)\}(\ell)$, 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

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 elliptic curves 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 Cramer-Shoup encryption in either of the two base groups. For completeness, we show key-verifiability of CS-encryption where keys are generated with respect to a common group description (including both generators) in App. C). 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 [AFG+10]. Moreover, the SPS (e.g., Groth [Gro15]) needs to be compatible with efficient labeled NIZK; the latter can be instantiated from standard Σ-protocols using the compiler by Faust et al. [FKMV12] and supporting labels is straightforward (cf. [ABM15]). As PCH instantiation we can use a strengthened version of the PCH by Derler et al. [DSSS19]. See App. C. To make the public key of the PCH compatible with the NIZK and Σ, it can simply be hashed using a collision-resistant hash-function.

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 NIZK. For sanitizing, the sanitizer has to create an encryption, adapt a hash, and attaches a simple NIZK. 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 NIZK. Likewise, proof-generation is a simple decryption and a NIZK proving that decryption was done honestly. Checking a proof is verifying a proof and a signature. Thus, ignoring the NIZK and the encryptions, our scheme is comparable to existing, way less expressive, constructions.

5 Conclusion

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. Our construction 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.

References

ABC$^+$15. Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. *J. Cryptology*, 28(2):351–395, 2015.

ABM15. Michel Abdalla, Fabrice Benhamouda, and Philip MacKenzie. Security of the J-PAKE password-authenticated key exchange protocol. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 571–587. IEEE Computer Society, 2015.

ACdMT05. Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, volume 3679 of *Lecture Notes in Computer Science*, pages 159–177. Springer, 2005.

ADK$^+$13. Masayuki Abe, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Tagged one-time signatures: Tight security and optimal tag size. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*, volume 7778 of *Lecture Notes in Computer Science*, pages 312–331. Springer, 2013.

AdM04. Giuseppe Ateniese and Breno de Medeiros. On the key exposure problem in chameleon hashes. In Carlo Blundo and Stelvio Cimato, editors, *Security in Communication Networks, 4th International Conference, SCN 2004, Amalfi, Italy, September 8-10, 2004, Revised Selected Papers*, volume 3352 of *Lecture Notes in Computer Science*, pages 165–179. Springer, 2004.

AFG$^+$10. Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 209–236. Springer, 2010.

AMVA17. Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton R. Andrade. Redactable blockchain - or - rewriting history in bitcoin and friends. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 111–126. IEEE, 2017.

BCD$^+$17. Michael Till Beck, Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Practical strongly invisible and strongly accountable sanitizable signatures. In Josef Pieprzyk and Suriadi Suriadi, editors, *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part I*, volume 10342 of *Lecture Notes in Computer Science*, pages 437–452. Springer, 2017.

BFF+09. Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In Jarecki and Tsudik [JT09], pages 317–336.

BFKW09. Dan Boneh, David Mandell Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Jarecki and Tsudik [JT09], pages 68–87.

BFLS09. Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Sanitizable signatures: How to partially delegate control for authenticated data. In Arslan Brömme, Christoph Busch, and Detlef Hühnlein, editors, *BIOSIG 2009 - Proceedings of the Special Interest Group on Biometrics and Electronic Signatures, 17.-18. September 2009 in Darmstadt, Germany*, volume P-155 of *LNI*, pages 117–128. GI, 2009.

BFLS10. Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2010.

BL17. Xavier Bultel and Pascal Lafourcade. Unlinkable and strongly accountable sanitizable signatures from verifiable ring signatures. In Srdjan Capkun and Sherman S. M. Chow, editors, *Cryptology and Network Security - 16th International Conference, CANS 2017, Hong Kong, China, November 30 - December 2, 2017, Revised Selected Papers*, volume 11261 of *Lecture Notes in Computer Science*, pages 203–226. Springer, 2017.

BLL+19. Xavier Bultel, Pascal Lafourcade, Russell W. F. Lai, Giulio Malavolta, Dominique Schröder, and Sri Aravinda Krishnan Thyagarajan. Efficient invisible and unlinkable sanitizable signatures. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part I*, volume 11442 of *Lecture Notes in Computer Science*, pages 159–189. Springer, 2019.

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

BPS12. Christina Brzuska, Henrich Christopher Pöhls, and Kai Samelin. Non-interactive public accountability for sanitizable signatures. In Sabrina De Capitani di Vimercati and Chris J. Mitchell, editors, *Public Key Infrastructures, Services and Applications - 9th European Workshop, EuroPKI 2012, Pisa, Italy, September 13-14, 2012, Revised Selected Papers*, volume 7868 of *Lecture Notes in Computer Science*, pages 178–193. Springer, 2012.

BPS13. Christina Brzuska, Henrich Christopher Pöhls, and Kai Samelin. Efficient and perfectly unlinkable sanitizable signatures without group signatures. In Sokratis K. Katsikas and Isaac Agudo, editors, *Public Key Infrastructures, Services and Applications - 10th European Workshop, EuroPKI 2013, Egham, UK, September 12-13, 2013, Revised Selected Papers*, volume 8341 of *Lecture Notes in Computer Science*, pages 12–30. Springer, 2013.

BPS17. Arne Bilzhause, Henrich C. Pöhls, and Kai Samelin. Position paper: The past, present, and future of sanitizable and redactable signatures. In *Proceedings of the 12th International Conference on Availability, Reliability* *and Security, Reggio Calabria, Italy, August 29 - September 01, 2017*, pages 87:1–87:9. ACM, 2017.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, pages 62–73. ACM, 1993.

BSW07. John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*, pages 321–334. IEEE Computer Society, 2007.

BSZ05. Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2005.

CDK+17. Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures. In Serge Fehr, editor, *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part II*, volume 10175 of *Lecture Notes in Computer Science*, pages 152–182. Springer, 2017.

CJ10. Sébastien Canard and Amandine Jambert. On extended sanitizable signature schemes. In Josef Pieprzyk, editor, *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings*, volume 5985 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2010.

CJL12. Sébastien Canard, Amandine Jambert, and Roch Lescuyer. Sanitizable signatures with several signers and sanitizers. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *Progress in Cryptology - AFRICACRYPT 2012 - 5th International Conference on Cryptology in Africa, Ifrance, Morocco, July 10-12, 2012. Proceedings*, volume 7374 of *Lecture Notes in Computer Science*, pages 35–52. Springer, 2012.

CLM08. Sébastien Canard, Fabien Laguillaumie, and Michel Milhau. Trapdoorsanitizable signatures and their application to content protection. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security, 6th International Conference, ACNS 2008, New York, NY, USA, June 3-6, 2008. Proceedings*, volume 5037 of *Lecture Notes in Computer Science*, pages 258–276, 2008.

CS97. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 1997.

CS98. Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998.

CvH91. David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.

DDH+15. Denise Demirel, David Derler, Christian Hanser, Henrich C. Pöhls, Daniel Slamanig, and Giulia Traverso. PRISMACLOUD D4.4: Overview of Functional and Malleable Signature Schemes. Technical report, H2020 Prismacloud, www.prismacloud.eu, 2015.

DHLW10. Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 613–631. Springer, 2010.

dMPPS14. Hermann de Meer, Henrich Christopher Pöhls, Joachim Posegga, and Kai Samelin. On the relation between redactable and sanitizable signature schemes. In Jan Jürjens, Frank Piessens, and Nataliia Bielova, editors, *Engineering Secure Software and Systems - 6th International Symposium, ESSoS 2014, Munich, Germany, February 26-28, 2014, Proceedings*, volume 8364 of *Lecture Notes in Computer Science*, pages 113–130. Springer, 2014.

DPSS15. David Derler, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. A general framework for redactable signatures and new constructions. In Soonhak Kwon and Aaram Yun, editors, *Information Security and Cryptology - ICISC 2015 - 18th International Conference, Seoul, South Korea, November 25-27, 2015, Revised Selected Papers*, volume 9558 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2015.

DS15. David Derler and Daniel Slamanig. Rethinking privacy for extended sanitizable signatures and a black-box construction of strongly private schemes. In Man Ho Au and Atsuko Miyaji, editors, *Provable Security - 9th International Conference, ProvSec 2015, Kanazawa, Japan, November 24-26, 2015, Proceedings*, volume 9451 of *Lecture Notes in Computer Science*, pages 455–474. Springer, 2015.

DS19. David Derler and Daniel Slamanig. Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge. *Des. Codes Cryptogr.*, 87(6):1373–1413, 2019.

DSSS19. David Derler, Kai Samelin, Daniel Slamanig, and Christoph Striecks. Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

FF15. Victoria Fehr and Marc Fischlin. Sanitizable signcryption: Sanitization over encrypted data (full version). *IACR Cryptology ePrint Archive*, 2015:765, 2015.

FH18. Marc Fischlin and Patrick Harasser. Invisible sanitizable signatures and public-key encryption are equivalent. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 202–220. Springer, 2018.

FKM$^+$16. Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, volume 9614 of *Lecture Notes in Computer Science*, pages 301–330. Springer, 2016.

FKMV12. Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the fiat-shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, volume 7668 of *Lecture Notes in Computer Science*, pages 60–79. Springer, 2012.

GGOT16. Esha Ghosh, Michael T. Goodrich, Olga Ohrimenko, and Roberto Tamassia. Verifiable zero-knowledge order queries and updates for fully dynamic lists and trees. In Vassilis Zikas and Roberto De Prisco, editors, *Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016, Proceedings*, volume 9841 of *Lecture Notes in Computer Science*, pages 216–236. Springer, 2016.

GQZ10. Junqing Gong, Haifeng Qian, and Yuan Zhou. Fully-secure and practical sanitizable signatures. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 6th International Conference, Inscrypt 2010, Shanghai, China, October 20-24, 2010, Revised Selected Papers*, volume 6584 of *Lecture Notes in Computer Science*, pages 300–317. Springer, 2010.

Gro06. Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*, pages 444–459. Springer, 2006.

Gro15. Jens Groth. Efficient fully structure-preserving signatures for large messages. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 239–259. Springer, 2015.

JMSW02. Robert Johnson, David Molnar, Dawn Xiaodong Song, and David A. Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference, 2002, San Jose, CA, USA, February 18-22, 2002, Proceedings*, volume 2271 of *Lecture Notes in Computer Science*, pages 244–262. Springer, 2002.

JT09. Stanislaw Jarecki and Gene Tsudik, editors. *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*, volume 5443 of *Lecture Notes in Computer Science*. Springer, 2009.

KPSS18a. Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Chameleon-hashes with dual long-term trapdoors and their applications. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings*, volume 10831 of *Lecture Notes in Computer Science*, pages 11–32. Springer, 2018.

KPSS18b. Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Protean signature schemes. In Jan Camenisch and Panos Papadimitratos, editors, *Cryptology and Network Security - 17th International Conference, CANS 2018, Naples, Italy, September 30 - October 3, 2018, Proceedings*, volume 11124 of *Lecture Notes in Computer Science*, pages 256–276. Springer, 2018.

KPSS19. Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Fully invisible protean signatures schemes. *IACR Cryptology ePrint Archive*, 2019:39, 2019.

KR00. Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA*, pages 143–154. The Internet Society, 2000.

KSS15. Stephan Krenn, Kai Samelin, and Dieter Sommer. Stronger security for sanitizable signatures. In Joaquín García-Alfaro, Guillermo Navarro-Arribas, Alessandro Aldini, Fabio Martinelli, and Neeraj Suri, editors, *Data Privacy Management, and Security Assurance - 10th International Workshop, DPM 2015, and 4th International Workshop, QASA 2015, Vienna, Austria, September 21-22, 2015. Revised Selected Papers*, volume 9481 of *Lecture Notes in Computer Science*, pages 100–117. Springer, 2015.

LDW13. Junzuo Lai, Xuhua Ding, and Yongdong Wu. Accountable trapdoor sanitizable signatures. In Robert H. Deng and Tao Feng, editors, *Information Security Practice and Experience - 9th International Conference, ISPEC 2013, Lanzhou, China, May 12-14, 2013. Proceedings*, volume 7863 of *Lecture Notes in Computer Science*, pages 117–131. Springer, 2013.

LOS+10. Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 62–91. Springer, 2010.

SBZ01. Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content extraction signatures. In Kwangjo Kim, editor, *Information Security and Cryptology - ICISC 2001, 4th International Conference Seoul, Korea, December 6-7, 2001, Proceedings*, volume 2288 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2001.

SS20. Kai Samelin and Daniel Slamanig. Policy-based sanitizable signatures. In Stanislaw Jarecki, editor, *Topics in Cryptology - CT-RSA 2020 - The*

Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings, volume 12006 of *Lecture Notes in Computer Science*, pages 538–563. Springer, 2020.

SSE+12. Yusuke Sakai, Jacob C. N. Schuldt, Keita Emura, Goichiro Hanaoka, and Kazuo Ohta. On the security of dynamic group signatures: Preventing signature hijacking. In Marc Fischlin, Johannes A. Buchmann, and Mark Manulis, editors, *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, volume 7293 of *Lecture Notes in Computer Science*, pages 715–732. Springer, 2012.

YAHK11. Shota Yamada, Nuttapong Attrapadung, Goichiro Hanaoka, and Noboru Kunihiro. Generic constructions for chosen-ciphertext secure attribute based encryption. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*, volume 6571 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2011.

YSL10. Dae Hyun Yum, Jae Woo Seo, and Pil Joong Lee. Trapdoor sanitizable signatures made easy. In Jianying Zhou and Moti Yung, editors, *Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings*, volume 6123 of *Lecture Notes in Computer Science*, pages 53–68, 2010.

A Additional Preliminaries

Definition 16 (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 17 (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:*

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

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

Fig. 13: Σ Unforgeability

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

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

The corresponding experiment is depicted in Figure 13.

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 19 (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-CCA2 secure and key-verifiable.

Definition 20 (Π IND-CCA2 Security). *An encryption scheme Π is IND-CCA2 secure, if for any PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

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

The corresponding experiment is depicted in Figure 14.

$\mathbf{Exp}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},\Pi}(\kappa)$
$\quad \mathsf{pp}_\Pi \leftarrow_r \mathsf{PPGen}_\Pi(1^\kappa)$
$\quad (\mathsf{sk}_\Pi, \mathsf{pk}_\Pi) \leftarrow_r \mathsf{KGen}_\Pi(\mathsf{pp}_\Pi)$
$\quad b \leftarrow_r \{0,1\}$
$\quad ((m_0^*, m_1^*), state_\mathcal{A}) \leftarrow_r \mathcal{A}^{\mathsf{Dec}_\Pi(\mathsf{sk}_\Pi,\cdot)}(\mathsf{pk}_\Pi)$
$\quad$ If $|m_0^*| \neq |m_1^*| \vee m_0^* \notin \mathcal{M} \vee m_1^* \notin \mathcal{M}$:
$\quad\quad c^* \leftarrow \perp$
$\quad$ Else:
$\quad\quad c^* \leftarrow_r \mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, m_b^*)$
$\quad b^* \leftarrow_r \mathcal{A}^{\mathsf{Dec}'_\Pi(\mathsf{sk}_\Pi,\cdot)}(state_\mathcal{A}, c^*)$
$\quad\quad$ where Dec'_Π on input sk_Π and c:
$\quad\quad\quad$ return $\perp$, if $c = c^*$
$\quad\quad\quad$ return $\mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, c)$
$\quad$ return 1, if $b^* = b$
$\quad$ return 0

Fig. 14: Π IND-CCA2 Security

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

Fig. 15: Π Key-Verifiability

Definition 21 (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 15.

Non-Interactive Proof Systems. Let L be an NP-language with associated witness relation R, i.e., such that $L = \{x \mid \exists w : R(x,w) = 1\}$. A non-interactive proof system allows to prove membership of some statement x in the language L. More formally, such a system is defined as follows.

Definition 22 (Non-Interactive Proof System). *A non-interactive proof system Ω for language L consists of three algorithms* $\{\mathsf{PPGen}_\Omega, \mathsf{Prove}_\Omega, \mathsf{Verify}_\Omega\}$*, such that:*

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

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

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

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

Verify_Ω**.** *The deterministic algorithm* Verify_Ω *verifies the proof π by outputting a bit $d \in \{0,1\}$, w.r.t. to some CRS* crs_Ω *and some statement statement x:*

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

Definition 23 (Completeness). *A non-interactive proof system is called complete, if for all $\kappa \in \mathbb{N}$, for all $\mathsf{crs}_\Omega \leftarrow_r \mathsf{PPGen}_\Omega(1^\kappa)$, for all $x \in L$, for all w such that $R(x,w) = 1$, for all $\pi \leftarrow_r \mathsf{Prove}_\Omega(\mathsf{crs}_\Omega, x, w)$, it holds that $\mathsf{Verify}_\Omega(\mathsf{crs}_\Omega, x, \pi) = 1$.*

In addition to completeness, we require two standard security notions for zero-knowledge proofs of knowledge: zero-knowledge and simulation-sound extractability. We define them analogous to the definitions given in [DS19].

Informally speaking, zero-knowledge says that the receiver of the proof π does not learn anything except the validity of the statement.

Definition 24 (Zero-Knowledge). *A non-interactive proof system Ω for language L is zero-knowledge, if for any PPT adversary $\mathcal{A}$, there exists an PPT simulator $\mathsf{SIM} = (\mathsf{SIM}_1, \mathsf{SIM}_2)$ such that there exist negligible functions ν_1 and ν_2 such that*

$$\Big| \Pr\left[\mathsf{crs}_\Omega \leftarrow_r \mathsf{PPGen}_\Omega(1^\kappa) \;:\; \mathcal{A}(\mathsf{crs}_\Omega) = 1\right] - \Pr\left[(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa) \;:\; \mathcal{A}(\mathsf{crs}_\Omega) = 1\right] \Big| \leq \nu_1(\kappa),$$

and that

$$\left| \Pr\left[\mathbf{Exp}^{\mathsf{Zero\text{-}Knowledge}}_{\mathcal{A},\Omega,\mathsf{SIM}}(\kappa) = 1\right] - \tfrac{1}{2} \right| \leq \nu_2(\kappa),$$

where the corresponding experiment is depicted in Figure 16.

$\mathbf{Exp}^{\mathsf{Zero\text{-}Knowledge}}_{\mathcal{A},\Omega,\mathsf{SIM}}(\kappa)$
$(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa)$
$b \leftarrow_r \{0,1\}$
$b^* \leftarrow_r \mathcal{A}^{P_b(\cdot,\cdot)}(\mathsf{crs}_\Omega)$
where P_0 on input x and w:
return $\pi \leftarrow_r \mathsf{Prove}_\Omega(\mathsf{crs}_\Omega, x, w)$, 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. 16: Ω Zero-Knowledge

Simulation-sound extractability says every adversary which is able to come up with a proof π^* for a statement must know the witness, even when seeing proofs for statements potentially not in L. Clearly, this implies that the proofs output by a simulation-sound extractable proof-systems are non-malleable. Note that the definition of simulation-sound extractability

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

Fig. 17: Ω Simulation-Sound Extractability

of [Gro06] is stronger than ours in the sense that the adversary also gets the trapdoor ζ as input. However, in our context this weaker notion (previously also used [ADK⁺13, DHLW10, DS19]) suffices.

Definition 25 (Simulation-Sound Extractability). *A zero-knowledge non-interactive proof system Ω for language L is said to be simulation-*

sound extractable, if for any PPT adversary $\mathcal{A}$, there exists a PPT extractor $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$, such that

$$\Big|\Pr\left[(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa) \ : \ \mathcal{A}(\mathsf{crs}_\Omega, \tau) = 1\right] - \Pr\left[(\mathsf{crs}_\Omega, \tau, \zeta) \leftarrow_r \mathcal{E}_1(1^\kappa) \ : \ \mathcal{A}(\mathsf{crs}_\Omega, \tau) = 1\right]\Big| = 0,$$

and that there exist a negligible function ν so that

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

where the corresponding experiment is depicted in Figure 17.

Supporting Labels. We note that to support labels, the definition of the Prove_Ω, Verify_Ω, SIM_2, and $\mathcal{E}_2$ algorithms also take a public label ℓ as input, and the completeness, soundness, and zero-knowlegde properties are updated accordingly (cf. [DHLW10]). Achieving this for SSE NIZK proofs obtained via the Fiat-Shamir transform from Σ-protocol [FKMV12] can be efficiently done by including the label into the hash computation (cf. [ABM15]).

B More Preliminaries and Building Blocks

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

B.1 Additional Preliminaries

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

Known-Order Group Definitions and Assumptions.

Group Generator. Let $(\mathbb{G}, g, q) \leftarrow_r \mathsf{DLGen}(1^\kappa)$ be a group-generator, where $\mathbb{G}$ is multiplicatively written and of prime-order q, where g is a generator.

Discrete Logarithm Assumption. Let $(\mathbb{G}, g, q) \leftarrow_r \mathsf{DLGen}(1^\kappa)$ be as defined above. The discrete logarithm assumption states that given g^x for some random $x \leftarrow_r \mathbb{Z}_q$, it is hard to find that x.

Definition 26 (Discrete Logarithm Assumption). *The discrete logarithm assumption holds, if for every PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\Pr[(\mathbb{G}, g, q) \leftarrow_r \mathsf{DLGen}(1^\kappa), x \leftarrow_r \mathbb{Z}_q, x' \leftarrow_r \mathcal{A}(\mathbb{G}, g, q, g^x) : x = x'] \leq \nu(\kappa)$$

Decisional Diffie-Hellman Assumption. Let $(\mathbb{G}, g, q) \leftarrow_r \mathsf{DLGen}(1^\kappa)$ be as defined above. The decisional Diffie-Hellman (DDH) assumption states that given (g^x, g^y, g^{xy}) is computationally indistinguishable from (g^x, g^y, g^z) for some random $x, y, z \leftarrow_r \mathbb{Z}_q^3$.

Definition 27 (Decisional Diffie-Hellman Assumption). *The decisional Diffie-Hellman assumption holds, if for every PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\Big| \Pr[(\mathbb{G}, g, q) \leftarrow_r \mathsf{DLGen}(1^\kappa), (x, y, z) \leftarrow_r \mathbb{Z}_q^3, b \leftarrow_r \{0,1\}$$
$$b' \leftarrow_r \mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^{bxy+(1-b)z}) : b = b'] - 1/2 \Big| \leq \nu(\kappa)$$

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 [BNPS03]. 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 $y_i \in \mathbb{Z}_n^*$, it is hard to invert more challenges than calls to the inversion oracle.

Definition 28 (One-More-RSA Inversion Assumption). *The one-more RSA inversion assumption holds, if 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.

B.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 [KR00].

The following framework is derived from Camenisch et al. [CDK+17].

Definition 29 (Chameleon-Hashes). *A chameleon-hash* CH *consists of five algorithms* $(\mathsf{PPGen}_{\mathsf{CH}}, \mathsf{KGen}_{\mathsf{CH}}, \mathsf{Hash}_{\mathsf{CH}}, \mathsf{Verify}_{\mathsf{CH}}, \mathsf{Adapt}_{\mathsf{CH}})$*, such that:*

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

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

$\mathsf{KGen}_{\mathsf{CH}}$. *The algorithm* $\mathsf{KGen}_{\mathsf{CH}}$*, given the public parameters* $\mathsf{pp}_{\mathsf{CH}}$*, outputs the private (*$\mathsf{sk}_{\mathsf{CH}}$*) and public key* $(\mathsf{pk}_{\mathsf{CH}})$ *of the scheme*

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

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

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

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

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

$\mathsf{Adapt}_{\mathsf{CH}}$. *The algorithm* $\mathsf{Adapt}_{\mathsf{CH}}$ *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'*:*

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

Note that we assume that the $\mathsf{Adapt}_{\mathsf{CH}}$ 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 $\mathsf{pp}_{\mathsf{CH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$, for all $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$, for all $m \in \mathcal{M}$, for all $(h, r) \leftarrow_r \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}, m)$, for all $m' \in \mathcal{M}$, we have for all for all $r' \leftarrow_r \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r, h)$, that $1 = \mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, h, r) = \mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m', h, r')$. This definition captures perfect correctness.

The randomness is drawn by $\mathsf{Hash}_{\mathsf{CH}}$, and not outside. This was done to capture "private-coin" constructions [AMVA17].

Next, we present security notions of CHs.

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

We relax the indistinguishability definition by Brzuska et al. [BFF+09] to a computational version, which is enough for most use-cases, including ours. However, compared to the existing definitions in [BCD+17, CDK+17, DSSS19, KPSS18a], the adversary is now also allowed to *generate* the secret keys involved.

$\mathbf{Exp}^{\mathsf{FIndistinguishability}}_{\mathcal{A},\mathsf{CH}}(\kappa)$
  $\mathsf{pp}_{\mathsf{CH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$
  $b \leftarrow_r \{0,1\}$
  $b^* \leftarrow_r \mathcal{A}^{\mathsf{HashOrAdapt}(\cdot,\cdot,\cdot,\cdot,b)}(\mathsf{pp}_{\mathsf{CH}})$
    where oracle $\mathsf{HashOrAdapt}$ on input $\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}, m, m', b$:
      $(h, r) \leftarrow_r \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m')$
      $(h', r') \leftarrow_r \mathsf{Hash}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m)$
      $r'' \leftarrow_r \mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r', h')$
      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 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 30 (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:*

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

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 [AdM04]: 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.

$\mathbf{Exp}_{\mathcal{A},\mathsf{CH}}^{\mathsf{Collision\text{-}Resistance}}(\kappa)$
  $\mathsf{pp}_{\mathsf{CH}} \leftarrow_r \mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)$
  $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$
  $\mathcal{Q} \leftarrow \emptyset$
  $(m^*, r^*, m'^*, r'^*, h^*) \leftarrow_r \mathcal{A}^{\mathsf{Adapt}'_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}},\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk}_{\mathsf{CH}})$
    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 31 (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:*

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

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 32 (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 [CDK+17]. 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 33 (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 [BCD+17, CDK+17, DSSS19]—the adversary can additionally generate all secret keys.

Definition 34 (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:*

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

The corresponding experiment is depicted in Figure 21.

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

Fig. 21: CHET Full Indistinguishability

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.

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

Fig. 22: CHET Public Collision-Resistance

Definition 35 (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.

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

Definition 36 (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:*

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

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^*.

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

Fig. 24: CHET Uniqueness

Definition 37 (CHET Uniqueness). *We say a* CHET *scheme is unique, if for every PPT adversary $\mathcal{A}$, there exists a negligible function ν such that:*

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

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) [BSW07].

Definition 38 (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 IND-CCA2 security, for ABE. It is derived from the definition given by Lewko et al. [LOS+10] and Derler et al. [DSSS19], but altered for our used notation. Refer, e.g., to [YAHK11] for how to construct chosen-ciphertext secure ABEs from CPA-secure ones.

Definition 39 (ABE **IND-CCA2-Security).** *An* ABE *scheme is IND-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.

C Concrete Instantiations of Primitives

We now present the instantiations of our building blocks.

Instantiation of Secure CHs. We recall a construction from [CDK+17] in Construction 2. In the construction, we assume that the size of e is always checked before usage.

Theorem 2. *If the one-more-RSA inversion assumption [BNPS03] holds, then the construction of a* CH *given in Construction 2 is fully indistinguishable, correct, unique and collision-resistant, in the random-oracle model [BR93].*

Proof. All properties, but full indistinguishability, have already been proven by Camenisch et al. [CDK+17]. Thus, it remains to prove full indistinguishability.

$\mathbf{Exp}_{\mathcal{A},\mathsf{ABE}}^{\mathsf{IND\text{-}CCA2}}(\kappa)$:
$(\mathsf{msk}_{\mathsf{ABE}}, \mathsf{mpk}_{\mathsf{ABE}}) \leftarrow_r \mathsf{PPGen}_{\mathsf{ABE}}(1^\kappa)$
$b \leftarrow_r \{0,1\}$
$\mathcal{Q} \leftarrow \emptyset$
$\mathcal{S} \leftarrow \emptyset$
$i \leftarrow 0$
$(m_0, m_1, \mathbb{A}^*, \mathtt{state}) \leftarrow_r \mathcal{A}^{\mathsf{KGen}'_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}},\cdot),\mathsf{KGen}''_{\mathsf{ABE}}(\mathsf{msk}_{\mathsf{ABE}},\cdot),\mathsf{Dec}'_{\mathsf{ABE}}(\cdot,\cdot)}(\mathsf{mpk}_{\mathsf{ABE}})$
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 $\perp$, if $(j, \mathsf{ssk}) \notin \mathcal{Q}$ for some ssk
return $\mathsf{Dec}_{\mathsf{ABE}}(\mathsf{ssk}, c)$
if $m_0 \notin \mathcal{M} \ \vee \ m_1 \notin \mathcal{M} \ \vee \ |m_0| \neq |m_1| \ \vee \mathbb{A}^* \cap \mathcal{S} \neq \emptyset$, let $c^* \leftarrow \perp$
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 $\perp$, 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 $\perp$, 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

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 $\rightarrow$ Game 1: As there is exactly randomness r (up to the group order, which can be ignored, as the verification algorithms check that case), which makes adaption work correctly, which we explicitly check, while r is always chosen uniformly random, 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.

$\underline{\mathsf{PPGen}_{\mathsf{CH}}(1^\kappa)}$: On input a security parameter κ it outputs the public parameters $\mathsf{pp}_{\mathsf{CH}} \leftarrow (1^\kappa, e)$, where e is prime and $e > N'$, and

$$N' = \max_r \{N \in \mathbb{N} : (N, \cdot, \cdot, \cdot, \cdot) \leftarrow_r \mathsf{RSAGen}(1^\kappa; r)\}$$

$\underline{\mathsf{KGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})}$: 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 $H : \{0,1\}^* \rightarrow \mathbb{Z}_N^*$ (modeled as a random-oracle), compute d s.t. $ed \equiv 1 \mod \varphi(N)$, set $\mathsf{sk}_{\mathsf{CH}} \leftarrow d$, $\mathsf{pk}_{\mathsf{CHET}} \leftarrow (N, H)$, and return $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}})$.

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

$\underline{\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, h, r)}$: 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 $h' \leftarrow H(m)r^e \mod N$ and outputs 1 if $h' = h$ and 0 otherwise.

$\underline{\mathsf{Adapt}_{\mathsf{CH}}(\mathsf{sk}_{\mathsf{CH}}, m, m', h, r)}$: 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 $\perp$ if $\mathsf{Verify}_{\mathsf{CH}}(\mathsf{pk}_{\mathsf{CH}}, m, h, r) \neq 1$. Otherwise, let $x \leftarrow H(m)$, $x' \leftarrow H(m')$, $y \leftarrow xr^e \mod N$. Output $\perp$, 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. [KPSS18a], 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.*

Proof. All properties, but full indistinguishability and uniqueness, have already been proven [DSSS19, KPSS18a]. 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

$\underline{\mathsf{PPGen}_{\mathsf{CHET}}(1^\kappa)}$: 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 $(\mathsf{sk}^1_{\mathsf{CH}}, \mathsf{pk}^1_{\mathsf{CH}}) \leftarrow_r \mathsf{KGen}_{\mathsf{CH}}(\mathsf{pp}_{\mathsf{CH}})$. Return $(\mathsf{sk}^1_{\mathsf{CH}}, \mathsf{pk}^1_{\mathsf{CH}})$.

$\underline{\mathsf{Hash}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m)}$: 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

$$((h^1, h^2, \mathsf{pk}^1_{\mathsf{CH}}, \mathsf{pk}^2_{\mathsf{CH}}), (r^1, r^2), \mathsf{etd})$$

$\underline{\mathsf{Verify}_{\mathsf{CHET}}(\mathsf{pk}_{\mathsf{CHET}}, m, h, r)}$: 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 $r = (r^1, r^2)$, 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

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 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 HashOrAdapt always checks if the adaption was successful, and thus so does $\mathcal{B}$, making the distributions equal. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{CH\text{-}FInd}}(\kappa)$ 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. $|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{CH\text{-}FInd}}(\kappa)$ 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 $(\mathsf{pk}'^*, (m^*, \mathsf{pk}_{\mathsf{CH}}^1, \mathsf{pk}_{\mathsf{CH}}^2), r_i^*, r_i'^*, h_i^*)$, where $\mathsf{pk}'^* = \mathsf{pk}_{\mathsf{CH}}^1$ if $i = 1$ and $\mathsf{pk}'^* = \mathsf{pk}_{\mathsf{CH}}^2$ 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 [DSSS19], but we also check whether an adaption was successful.

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.*

$\underline{\mathsf{PPGen}_{\mathsf{PCH}}(1^\kappa)}$: 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

Note, we do not require outsider collision-resistance. However, this property was already proven by Derler et al. [DSSS19].

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 $(\mathsf{pk}_1^*, m^*, r^*, r'^*, h_0^*)$, 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 $\mathsf{HashOrAdapt}$ 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 $\mathsf{HashOrAdapt}$ always checks if the adaption was successful, and thus so does $\mathcal{B}$, making the output behave the same. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\mathsf{CHET\text{-}FInd}}(\kappa)$ 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 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 $i \leftarrow_r \{1, 2, \ldots, q\}$. 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 $i \leftarrow_r \{1, 2, \ldots, q\}$. 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 Cramer and Shoup [CS98] in Construction 5, with the alteration that g_1 and g_2 are part of the parameters. We require this alteration to prove key-verifiability.

$\underline{\mathsf{PPGen}_\Pi(1^\kappa)}$: On input a security parameter κ, it outputs the public parameters $\mathsf{pp}_\Pi = (\mathbb{G}, g_1, g_2, q, \mathsf{H})$, where $(\mathbb{G}, g, q) \leftarrow_r \mathsf{DLGen}(1^\kappa)$, where g is some generator of $\mathbb{G}$. Draw $x \leftarrow_r \mathbb{Z}_q$, and let $g_1 \leftarrow g$ and $g_2 \leftarrow g^x$. H is some universal hash-function.

$\underline{\mathsf{KGen}_\Pi(\mathsf{pp}_\Pi)}$: On input $\mathsf{pp}_\Pi = (\mathbb{G}, g_1, g_2, q, \mathsf{H})$, draw $(x_1, x_2, y_1, y_2, z) \leftarrow_r \mathbb{Z}_q^5$. Let $c \leftarrow g_1^{x_1} g_2^{x_2}$, $d \leftarrow g_1^{y_1} g_2^{y_2}$, and $h \leftarrow g_1^z$. Set $\mathsf{pk}_\Pi \leftarrow (g_1, g_2, c, d, h)$, and $\mathsf{sk}_\Pi \leftarrow (x_1, x_2, y_1, y_2, z)$. Return $(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi)$.

$\underline{\mathsf{Enc}_\Pi(\mathsf{pk}_\Pi, m)}$: On input a public key $\mathsf{pk}_\Pi = (g_1, g_2, c, d, h)$ and a message m, draw $r \leftarrow_r \mathbb{Z}_q$. Compute $u_1 \leftarrow g_1^r$, $u_2 \leftarrow g_2^r$, $e \leftarrow h^r m$, $\alpha \leftarrow \mathsf{H}(u_1, u_2, e)$, and $v \leftarrow c^r d^{r\alpha}$. Return (u_1, u_2, e, v).

$\underline{\mathsf{Dec}_\Pi(\mathsf{sk}_\Pi, c)}$: On input secret key $\mathsf{sk}_\Pi = (x_1, x_2, y_1, y_2, z)$, and a ciphertext $c = (u_1, u_2, e, v)$, compute $\alpha \leftarrow \mathsf{H}(u_1, u_2, e)$. If $u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha} \neq v$, return $\perp$. Return e/u_1^z.

$\underline{\mathsf{KVrf}_\Pi(\mathsf{sk}_\Pi, \mathsf{pk}_\Pi)}$: On input a secret key $\mathsf{sk}_\Pi = (x_1, x_2, y_1, y_2, z)$ and a public key $\mathsf{pk}_\Pi = (g_1, g_2, c, d, h)$, output 1, if $\mathsf{pk}_\Pi = (g_1^{x_1} g_2^{x_2}, g_1^{y_1} g_2^{y_2}, g_1^z)$, and 0 otherwise.

Construction 5: Cramer Shoup Π

Theorem 5. *If the DDH-Assumption holds in $\mathbb{G}$, then the above construction is correct, IND-CCA2 secure, and key-verifiable.*

Proof. Correctness and IND-CCA2 security have already been proven by Cramer and Shoup [CS98].

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. We can then construct an adversary $\mathcal{B}$ which breaks the discrete logarithm assumption. $\mathcal{B}$ proceeds as follows. It receives $(\mathbb{G}, g, q)$ and g^x. It embeds g as g_1 and g^x as g_2. Whenever $\mathcal{A}$ outputs $(x_1, x_2, y_1, y_2, z) \neq (x'_1, x'_2, y'_1, y'_2, z')$. We know that $z = z'$ must hold, as we are working in prime-order groups. Assume that $x_1 \neq x'_1$. It follows that $x_2 \neq x'_2$. $\mathcal{B}$ can extract x by calculating $x \leftarrow (x_1 - x'_1)/(x'_2 - x_2)$. The case that $y_1 \neq y'_1$ is similar. $|\Pr[S_0] - \Pr[S_1]| \leq 2\nu_{\mathsf{dlog}}(\kappa)$ follows.

As the adversary has no more possibilities to win the game, key-verifiability is proven.

D Proof of Theorem 1

We now present the proof of the Theorem 1.

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

Unforgeability. To prove 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 $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa)$, keep the trapdoor τ, and start simulating all proofs.

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 $(\mathsf{crs}_\Omega, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa)$ 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 was able to generate a signature σ_m^* on a string never generated by the signing-oracle. 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 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 2. 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_3 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. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{eUNF\text{-}CMA}}(\kappa)$ follows.

Game 4: As Game 3, but we abort, if the adversary was able to generate (m^*, σ^*) for which m^* should not have been derivable. 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 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 3. 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). $|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{PBCH\text{-}SInsider\text{-}CollRes}}(\kappa)$ follows.

Game 5: As Game 4, but we abort, if the adversary was able to generate (m^*, σ^*), but has never made a call to $\mathsf{AddSan}'_{\mathsf{P3S}}$. Let this event be E_5.

Transition - Game 4 → Game 5: Assume, towards contradiction, that event E_5 happens. We can then construct an adversary $\mathcal{B}$ which breaks the unforgeability of the used Σ or the one-wayness of the used one-way function f. Namely, $\mathcal{B}$ receives pk_Σ of the Σ and f, and $f(x) = y$ from its own challenger. This is embedded in $\mathsf{pk}_{\mathsf{P3S}}$ (and, of course, the public parameters), while all other values are generated as in Game 4. y is embedded in $\mathsf{pk}^{\mathsf{Sig}}_{\mathsf{P3S}}$. For signing, the proofs are already simulated, and thus x is not required to be known. Each call to $\mathsf{AddSan}_{\mathsf{P3S}}$ for keys for which the adversary knows the corresponding secret keys, $\mathcal{B}$ calls its signature oracle to obtain such a key. For simulated sanitizers, those signature do not need to be obtained, as the proofs are already simulated. Then, whenever the adversary outputs (m^*, σ^*), $\mathcal{B}$ extracts values $(x_1, x_2, \mathsf{sk}_\Pi, \sigma')$. If $f(x_1) = y$, $\mathcal{B}$ can return x_1 to break the one-wayness of f. In the other case, $\mathcal{B}$ can return $((f(x_2), \mathsf{pk}_{\mathsf{P3S}}), \sigma')$ as its own forgery attempt for Σ. If extraction fails or a wrong statement was proven, SSE does not hold. A reduction is straightforward.
$|\Pr[S_4] - \Pr[S_5]| \leq \nu_{\mathsf{eUNF\text{-}CMA}}(\kappa) + \nu_{\mathsf{ow}}(\kappa) + \nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ 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 $(\mathsf{crs}_{\Omega}, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa)$, keep the trapdoor τ, and start simulating all proofs.

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 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_2.

Transition - Game 1 → Game 2: Assume, towards contradiction, that event E_2 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 2. 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 $\mathsf{pk}_{\mathsf{P3S}}$.

$|\Pr[S_1] - \Pr[S_2]| \leq \nu_{\mathsf{PCH\text{-}uniq}}(\kappa)$ follows, while the signature does not matter, as it is already hidden behind a simulated zero-knowledge proof, making the distributions equal.

Game 3: As Game 2, 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 $\mathsf{pk}_{\mathsf{P3S}}^{\mathsf{San}}$, 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 2 → Game 3: 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 $\mathcal{B}$. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{PCH\text{-}FInd}}(\kappa)$ 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 $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa)$, keep the trapdoor τ, and start simulating all proofs.

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 the contents of c (or c' resp.) with a 0, if generated by the $\mathsf{SignOrSanit}$-oracle.

Transition - Game 1 → Game 2: Assume, towards contradiction, that the adversary behaves noticeably different. We can then construct an adversary $\mathcal{B}$ which breaks the IND-CCA2 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 1. 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 2.0 is the same as Game 1, while in Game 2.1., however, we make the first replacement. Then, whatever $\mathcal{A}$ outputs in Game 3.i is also output by $\mathcal{B}$. Note, if a ciphertext is to be decrypted (e.g., for proof-generation of signatures not generated by the $\mathsf{SignOrSanit}$-oracle), $\mathcal{B}$ uses the provided decryption oracle provided.

$|\Pr[S_1] - \Pr[S_2]| \leq q\nu_{\mathsf{ind\text{-}cca2}}(\kappa)$ 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 3: As Game 2, 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 2 → Game 3: 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 2. 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 $\mathcal{B}$. $|\Pr[S_2] - \Pr[S_3]| \leq \nu_{\mathsf{PCH\text{-}FInd}}(\kappa)$ 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 ($b = 1$). 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 $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa)$, keep the trapdoor τ, and start simulating all proofs.

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 $(\mathsf{crs}_\Omega, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa)$ 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 2. 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 replace the contents of c' with a 0, if generated by the $\mathsf{LoRSanit}$-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-CCA2 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. Thus, Game 4.0 is the same as Game 4, while in Game 4.1., however, we make the first replacement. Then, whatever $\mathcal{A}$ outputs in Game 4.i is also output by $\mathcal{B}$. All decryption queries required can be obtained by the decryption oracle provided. $|\Pr[S_3] - \Pr[S_4]| \leq q\nu_{\textsf{ind-cca2}}(\kappa)$ 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 $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa)$, keep the trapdoor τ, and start simulating all proofs.

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 $\mathcal{B}$. $|\Pr[S_0] - \Pr[S_1]| \leq \nu_{\textsf{nizk-zk}}(\kappa)$ 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 $(\mathsf{crs}_\Omega, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa)$ 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 extract $(x_1, x_2, \mathsf{sk}_\Pi, r, \sigma_{\mathsf{sk}_\mathbb{S}})$ from π^*_Σ (the proof contained in σ^*), and sk'_Π from π^* (the one used by judge), if the winning conditions are met. Note, π^*_Σ is fresh or the statement is fresh, and thus the proof is not simulated. We abort, if extraction fails. Let this event be E_3.

Transition - Game 2 → Game 3: Assume, towards contradiction, that E_3 happened. We can then construct an adversary $\mathcal{B}$ against simulation-sound extractability of the used proof system. Namely, $\mathcal{B}$ receives crs_Ω from its own challenger, and embeds them in the public parameters. Proofs are simulated by the provided simulator. $|\Pr[S_2] - \Pr[S_3]| \leq 2\nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ follows.

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 or the key-verifiability of the used encryption scheme. The reduction works as follows. It receives f and $f(x)$ and the public parameters pp_Π. It embeds all values accordingly. Note, the proofs are simulated, and thus x is not needed to be known. Every sanitization is done honestly, with the exception of simulated proofs. Then, as we know that the adversary wins its game, $\mathcal{B}$ has either extracted $(x_1, \perp, \mathsf{sk}_\Pi, r, \perp)$ or $(\perp, x_2, \perp, r, \sigma_{\mathsf{sk}_\mathbb{S}})$ along with sk'_Π. In the case the reduction extracted $(\perp, x_2, \perp, r, \sigma_{\mathsf{sk}_\mathbb{S}})$, we can directly return x_2 to the one-way challenger. In the other case, the adversary found $\mathsf{sk}'_\Pi \neq \mathsf{sk}_\Pi$, as the decryption of c^* (contained in σ^*) decrypts to different plaintexts. Thus, the reduction can return $(\mathsf{sk}_\Pi, \mathsf{sk}'_\Pi, \mathsf{pk}_\Pi)$ as its own forgery.
$|\Pr[S_3] - \Pr[S_4]| \leq \nu_{\mathsf{key-verf}}(\kappa) + \nu_{\mathsf{owf}}(\kappa)$ 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 $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa)$, keep the trapdoor τ, and start simulating all proofs.

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 $\mathcal{B}$. $|\Pr[S_0] - \Pr[S_1]| \le \nu_{\mathsf{nizk\text{-}zk}}(\kappa)$ 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 $(\mathsf{crs}_\Omega, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa)$ 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 $(\mathsf{pk}^*, \sigma^*, m^*, \pi^*)$ such that the winning conditions are met. Let this event be E_3.

Transition - Game 2 → Game 3: Assume, towards contradiction, that E_3 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' (along with sk') such that $f(x') = f(x)$ (if extraction fails, this adversary breaks SSE using a straightforward reduction), and can return it to its own challenger. $|\Pr[S_2] - \Pr[S_3]| \le \nu_{\mathsf{owf}}(\kappa) + \nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ 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 $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa)$, 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, as we do not have any oracles. $|\Pr[S_0] - \Pr[S_1]| \le \nu_{\mathsf{nizk\text{-}zk}}(\kappa)$ follows.

Game 2: As Game 1, but we replace crs_{Ω} with the one generated by $(\mathsf{crs}_{\Omega}, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa)$ 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 $((\mathsf{pk}_i^*)_{0\le i\le 5}, \sigma^*, m^*, \pi_0^*, \pi_1^*)$ such that $\mathsf{pk}_2^* \neq \mathsf{pk}_5^*$, but $\mathsf{pk}_1^* = \mathsf{pk}_4^*$, while the winning conditions are met (Note, decryption is deterministic). 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 $((\mathsf{pk}_i^*)_{0\le i\le 5}, \sigma^*, m_0^*, m_1^*, \pi_0^*, \pi_1^*)$, $\mathcal{B}$ extracts sk_0^* from π_0^* and sk_1^* from π_1^* (if extraction fails, this adversary breaks SSE; a reduction is straightforward). Then, it can return $(\mathsf{sk}_0^*, \mathsf{sk}_1^*, \mathsf{pk}_1^*)$ as its own forgery.
$|\Pr[S_2] - \Pr[S_3]| \le \nu_{\mathsf{enc\text{-}key\text{-}verf}}(\kappa) + 2\nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ immediately follows.
This hop essentially rules out the possibility an adversary can create more than one secret key w.r.t. to its pk such that decryption points to a different sanitizer.

Game 4: As Game 4, but we abort if the adversary outputs $((\mathsf{pk}_i^*)_{0\le i\le 5}, \sigma^*, m_0^*, m_1^*, \pi_0^*, \pi_1^*)$ 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 $(\mathsf{pk}_i^*)_{0\le i\le 2}$) or the second statement (concerning $(\mathsf{pk}_i^*)_{3\le i\le 5}$), with the "proof" π contained in σ^*. (Note, all keys are part of the label, and we have assumed that they are non-malleable attached to each proof π). Which proof is wrong can easily be derived from the attached values. $|\Pr[S_3] - \Pr[S_4]| \le \nu_{\mathsf{nizk\text{-}sse}}(\kappa)$ 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 $(\mathsf{crs}_\Omega, \tau) \leftarrow_r \mathsf{SIM}_1(1^\kappa)$, keep the trapdoor τ, and start simulating all proofs.

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 $(\mathsf{crs}_\Omega, \tau, \xi) \leftarrow_r \mathcal{E}_1(1^\kappa)$ 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 $\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{Sig}}$) calculate 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.

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* 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 $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c)$, but $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c, 0)$. For sanitization, the label is changed to $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c, 1)$. 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 $\mathsf{sk}_{\mathsf{P3S}}^{\mathsf{Sig}}$, 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 $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c, c')$, 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, in the altered construction 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 $\mathsf{pk}_{\mathsf{P3S}}$ 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* 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 NIZK, 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 NIZK 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* 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 $\mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}$ are used, the last bit is simply dropped for the underlying algorithms. For the judge, however, if $\mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}$ 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 $\mathsf{pk}_{\mathsf{P3S}}^{\mathsf{San}}$. If some of the inner keys of $\mathsf{pk}_{\mathsf{P3S}}^{\mathsf{San}}$ are used, the last bit is simply dropped for the underlying algorithms. For the judge, however, if $\mathsf{pk}_{\mathsf{P3S}}^{\mathsf{San}}$ 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 $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c)$, but $\ell = (\mathsf{pp}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}, \mathsf{pk}_{\mathsf{P3S}}^{\mathsf{Sig}}, h, r, m, \mathsf{A}, \mathbb{A}, m_{\mathsf{A}}, m_{!\mathsf{A}}, \sigma_m, c, 0)$. 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.