# Lattice-Based Sanitizable Signature Schemes: Chameleon Hash Functions and More

Sebastian Clermont[1], Samed Düzlü[2], Christian Janson[1], Laurens Porzenheim[3], and Patrick Struck[4]

[1] Technische Universität Darmstadt, Germany
sebastian.clermont@tu-darmstadt.de christian.janson@cryptoplexity.de
[2] Universität Regensburg, Germany
samed.duzlu@ur.de
[3] Universität Paderborn, Germany
laurens.porzenheim@uni-paderborn.de
[4] Universität Konstanz, Germany
patrick.struck@uni-konstanz.de

**Abstract.** Sanitizable Signature Schemes (SSS) enable a designated party, the sanitizer, to modify predefined parts of a signed message without invalidating the signature, making them useful for applications like pseudonymization and redaction. Since their introduction by Ateniese et al. (ESORICS'05), several classical SSS constructions have been proposed, but none have been instantiated from quantum-resistant assumptions. In this work, we develop the first quantum-secure sanitizable signature schemes based on lattice assumptions. Our primary focus is on SSS constructions that rely on chameleon hash functions (CHFs), a key component for enabling the controlled modification of messages. While lattice-based CHFs exist, they do not meet the required security guarantees for SSS, becoming insecure under adversarial access to an adapt oracle. To address this, we construct a novel lattice-based CHF that achieves collision resistance even in such settings, called full collision resistance. However, our CHF lacks the *uniqueness* property, a limitation we show to be inherent in lattice-based CHFs. As a result, our SSS constructions initially fall short of achieving the critical security property of accountability. To overcome this, we apply a transformation based on verifiable ring signatures (VRS), for which we present the first lattice-based instantiation. Additionally, we provide a comprehensive analysis of existing classical SSS constructions, explore their potential for post-quantum instantiations, and present new attacks on previously assumed secure SSS schemes. Our work closes the gap in constructing quantum-secure SSS and lays the groundwork for further research into advanced cryptographic primitives based on lattice assumptions.

* This work was funded by the German Federal Ministry of Education and Research (BMBF) under reference 16KISQ074, the German Research Foundation (DFG) – SFB 1119 – 236615297, the Ministry of Culture and Science of the State of North Rhine-Westphalia, and the Hector Foundation II.
An extended abstract of this work was first published in PQCrypto 2025, LNCS 15577, pp. 278–311, 2025. https://doi.org/10.1007/978-3-031-86599-2_10

## 1 Introduction

In modern cryptography, the digital signature scheme is an essential primitive that enjoys ubiquitous usage. However, depending on the use case, one might need more advanced signature schemes which come with additional functionality. These advanced variants usually build upon their base versions, offering strong security guarantees for their specialized functionality and alleviating the need for ad-hoc constructions. Advanced signature-based primitives include sanitizable signatures, ring signatures, group signatures, threshold signatures, and more.

The main focus of this paper are sanitizable signature schemes (SSS). They can be applied, for instance, to pseudonymize sensitive data, like patient names in medical records, while maintaining the authenticity. Sanitizable signature schemes allow the signer of a message to partially delegate signature rights to a semi-trusted third party, the sanitizer. This sanitizer may modify predefined parts of the message without invalidating the signature. In a sanitizable signature scheme, the message is divided into blocks, among which the sanitizer can modify exactly the *admissible blocks*. In contrast, all other blocks, the *immutable blocks*, cannot be modified by the sanitizer without invalidating the signature. The signer defines which blocks are admissible during the signing operation. Another use-case of sanitizable signature schemes are classified documents. SSS constructions allow the signing of such documents (by the signer) such that a censor (the sanitizer) can redact certain parts, depending on which parts should not be made public.

Sanitizable signature schemes were first introduced in Ateniese et al.'s seminal work [3]. Since then, many more works on sanitizable signature schemes have been published, covering new constructions focusing on different, sometimes novel, security notions. Current literature contains only two constructions that exclusively use basic primitives [13, 22], one construction is based on group signatures [12], two are based on ring signatures [15, 30], one is based on signatures with rerandomizable keys [23], while all others, [11, 3, 30, 16, 6], make use of so-called *chameleon hash functions* (CHFs) [28].

Chameleon hash functions are families of hash functions with public and secret keys. In contrast to regular hash functions, their input consists of not just the message, but also an auxiliary random value. The public key of a chameleon hash function defines a collision-resistant function mapping pairs $(\mu, r)$ to some hash value $h$, where $\mu$ is the message and $r$ is a randomly chosen auxiliary value. With the corresponding secret key, however, one can then *adapt* the message to a new $\mu'$, by finding a suitable auxiliary value $r'$ such that the pair $(\mu', r')$ evaluates to the original digest $h$, i.e., find a collision. Besides weak collision resistance, where the adversary only receives the public key, chameleon hash functions can support stronger forms of collision resistance, where an adapt oracle is available to the adversary. Apart from SSS, chameleon hash functions are a widely used building block in other cryptographic schemes. For example, they can transform a non-adaptively secure signature scheme to an existentially unforgeable one [41].

The gist of constructing a SSS from a CHF is a simple hash-then-adapt approach. In this approach, the admissible blocks are hashed using the CHF

with a public key generated by the sanitizer and then signed by the signer using a regular signature scheme. The sanitizer can thus modify the message without altering the hash value, retaining the original signature's validity. More advanced constructions of SSS have been developed over the years, some of which attain additional security properties and explore alternative approaches.

Given the general transition to post-quantum cryptography, analyzing which advanced primitives can also be constructed from quantum-hard assumptions is crucial. While there are positive results for various advanced primitives, sanitizable signatures from quantum-hard assumptions have yet to be considered.

### 1.1 Our Contribution

We close the aforementioned gap by developing post-quantum secure sanitizable signatures from lattices. Our main focus is on constructions based on chameleon hash functions. While lattice-based CHFs exist [18, 31], we observe that they cannot be used to instantiate existing SSS constructions. The reason is that they do not achieve the necessary security guarantees. Thus, we develop a new chameleon hash function inspired by the one given in [31]. Our construction achieves collision resistance even against adversaries with access to an adapt oracle that provides collision to the adversary—for SSS constructions this is necessary since sanitized signatures are effectively that. This is our first contribution as we believe this strongly secure CHF can be of independent interest.

However, this construction is not sufficient to already instantiate the SSS constructions. The reason is that it lacks another property called *uniqueness*. When a chameleon hash has uniqueness, it is hard to come up with two different auxiliary values $r, r'$ that both map some message $\mu$ to the same digest $h$. Unfortunately, the lack of the uniqueness property seems not to be a flaw of our specific construction but a more general problem of lattice-based CHFs. The absence of this property implies that the SSS constructions lack crucial security properties like unforgeability and accountability. However, there is a generic transform [15] which turns such a SSS (which they call weakly-secure) into a secure SSS. This transform requires another advanced signature, called verifiable ring signature (VRS), i.e., a variant of a ring signature that allows to prove or deny ownership of a signature at a later point. Thus, as our second contribution, we develop a lattice-based VRS scheme. To do that, we first design a generic construction of a VRS scheme using a so-called linking indistinguishable tag and a non-interactive zero-knowledge proof. Then, we instantiate that generic construction with lattice-based constructions.

With these contributions (illustrated in Fig. 1), we close the aforementioned gap by giving multiple post-quantum sanitizable signatures from lattices.

As additional contributions, we give an attack against an SSS construction [11], breaking the accountability, i.e., an attack that allows the signer to blame the sanitizer for messages of its own choosing. Furthermore, we revisit the SSS constructions that do not rely on chameleon hash functions. We show that for some of them, the necessary building blocks are readily available in the lattice-based literature, meaning they can easily be instantiated as well. Finally, we

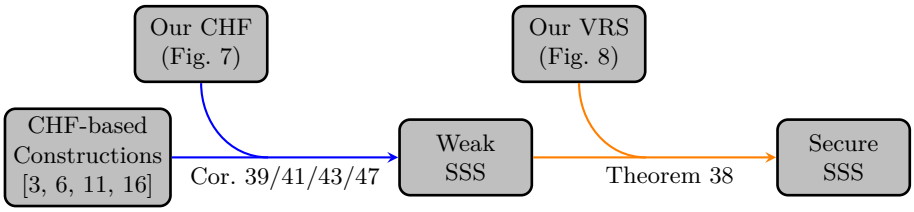


Fig. 1: Overview of our main results. Instantiating the SSS constructions given in [3, 6, 11, 16] with our new lattice-based chameleon hash function results in weak sanitizable signatures. The transformation from [15]—using these weak sanitizable signatures and our new verifiable ring signature scheme—then yields secure sanitizable signatures.

show additional implications between the security notions, which also show some additional security properties for some SSS construction—in particular, security properties that were developed after some of the older SSS constructions.

### 1.2 Related Work

Current literature provides a variety of constructions of sanitizable signature schemes. Two of them, [13, 22], require standard assumptions only. Some of them, [12, 15, 23] provide constructions based on other advanced signatures. The work [30] by Lai et al. contains two constructions: one based on accountable ring signatures and another one based on chameleon hash functions. Finally, many more constructions based on chameleon hash functions exist, [3, 11, 6, 16, 15]. Besides the different primitives they use, the various works have distinct goals regarding the security notions they achieve. The achieved security properties can be seen in Table 1. Another construction is due to Klonowski and Lauks [27]. Later, however, Canard and Jambert [17] showed it to lack accountability.

Constructions of chameleon hash functions based on lattices are very rare. Cash et al. [18] came forward with the first construction based on a trapdoor in the Ajtai hash function. This chameleon hash function achieves merely a weak form of collision-resistance. Only recently, a new lattice-based construction was brought forward by Li and Liu [31]. They use one-time tags, making the hash function not applicable to known generic constructions of SSS. However, we use and improve their ideas to construct our novel chameleon hash function.

Due to the missing uniqueness of our CHF, we apply the transform introduced in [15] to increase the security of the instantiations from weak SSS to strong SSS. The transform relies on verifiable ring signatures (VRS). In [15], it is stated that other ring signatures with extended functionality, such as linkable ring signatures, imply VRS. However, we were not able to verify that claim.

[39] present (among others) a notion of ring signatures equivalent to VRS, called claimable and repudiable ring signatures. They also show how to construct such a ring signature. However that requires verifiable random functions, which can be difficult to construct (e.g., [21]) or use (non-interactive) proofs in their

Table 1: Overview of our results. The table provides an overview of the SSS constructions that use chameleon hash functions: ACMT05 [3], BCD$^+$17 [6], CDK$^+$17 [16], and BFF$^+$09 [11]. The entries with the color blue are the results of the instantiations of the respective constructions with our chameleon hash function defined in Section 3.1. The entries with the color orange are the results after using our verifiable ring signature defined in Section 3.2 to apply a transform as described in [15] to ensure unforgeability as well as signer and sanitizer accountability. The various symbols represent: the instantiation achieves the notion (✓), the construction does not achieve the notion (✗), the instantiation does not achieve a notion that the construction otherwise could achieve (✦), the instantiation has not been analyzed with respect to the notion (**?**), and there is no lattice-based instantiation due to missing building blocks (❍). Note that some notions contradict each other, consequently it is impossible to come up with a construction achieving all notions (cf. Appendix B.2).

| | Chameleon Hash Function based Sanitizable Signatures | | | | |
|---|---|---|---|---|---|
| | ACMT05 [3] | LZCS16−1 [30] | BCD$^+$17 [6] | CDK$^+$17 [16] | BFF$^+$09 [11] |
| Unforgeability | ✓ | ❍\|✓ | ✦\|✓ | ✦\|✓ | ✓ |
| Signer Accountability | ✓ | ❍\|✓ | ✦\|✓ | ✓ | ✗\|✓ |
| Sanitizer Accountability | ?\|✓ | ❍\|✓ | ✦\|✓ | ✦\|✓ | ✓ |
| NIPA | ✗ | ✗ | ✗ | ✗ | ✗ |
| Immutability | ✓ | ❍ | ✓ | ✓ | ✓ |
| Transparency | ✓ | ❍ | ✓ | ✓ | ✓ |
| Privacy | ✓ | ❍ | ✓ | ✓ | ✓ |
| Unlinkability | ✗ | ✗ | ✗ | ✗ | ✗ |
| Invisibility | ? | ? | ✓ | ✓ | ? |

construction. Furthermore, the construction of [39] requires a proof over statements including the VRF, i.e., proofs over proofs, which is why we opted for a more direct approach for constructing a VRS.

## 2 Background

We define some notation. By $x \leftarrow y$ we define deterministic assignment of $y$ to $x$. If $S$ is some set, we denote by $x \leftarrow\!\!\$\ S$ sampling a uniform value $x$ from $S$. Overloading notation, if $A$ is some probabilistic polynomial-time (*ppt*) or quantum polynomial time (*qpt*) algorithm, we denote by $y \leftarrow\!\!\$\ A(x)$ assigning the random output of $A(x)$ to $y$. The statement of the kind "for all $y \leftarrow\!\!\$\ A(x)$" means for all $y$ that can be output by the randomized algorithm $A$ on input $x$. We denote column vectors by lowercase bold-face letters $\mathbf{a}$, while row-vectors are denoted as $\mathbf{a}^t$. Matrices $\mathbf{A}$ are denoted in uppercase. By $\|\mathbf{x}\|$ we denote the Euclidian norm of $\mathbf{x}$. If we want to query the random oracle on some input $x$, we

denote it by $y \leftarrow \mathcal{RO}(x)$. We will often assume security parameters and public parameters as implicit inputs to algorithms and omit them for readability.

The security analysis under the presence of quantum adversary takes place in the Q1 security model, allowing local quantum computer access to adversaries, but only classical access to oracles including the random oracle. Signature schemes as well as public key encryption schemes follow the commonly established syntax and notation, but for the sake of completeness are defined in Appendix A. We will formalize less ubiquitous primitives in this section.

### 2.1 Chameleon Hash Function

First, we introduce chameleon hash functions, which we will later build from lattices as one of our core contributions in this paper.

**Definition 1 (Chameleon Hash Function).** *A chameleon hash function* $\mathsf{CH}$ *consists of four ppt algorithms* $\mathsf{CH} = (\mathsf{CKGen}, \mathsf{CHash}, \mathsf{CHashCheck}, \mathsf{CAdapt})$.

$(\mathsf{pk_{ch}}, \mathsf{sk_{ch}}) \leftarrow\!\!\$\ \mathsf{CKGen}(1^\lambda)$: *The key generation algorithm* $\mathsf{CKGen}$ *takes as input the security parameter* $1^\lambda$ *and outputs a private-public key pair.*

$(h, r) \leftarrow\!\!\$\ \mathsf{CHash}(\mathsf{pk_{ch}}, \mu)$: *The algorithm* $\mathsf{CHash}$ *takes as input a public key* $\mathsf{pk_{ch}}$ *and a message* $\mu$*. It outputs a hash* $h$ *and randomness* $r$*, under the given public key* $\mathsf{pk_{ch}}$*.*

$b \leftarrow \mathsf{CHashCheck}(\mathsf{pk_{ch}}, h, \mu, r)$: *Given a public key* $\mathsf{pk_{ch}}$*, a hash value* $h$*, a message* $\mu$*, and randomness* $r$*, the deterministic algorithm* $\mathsf{CHashCheck}$ *outputs a boolean value* $b \in \{0, 1\}$*, indicating whether the hash* $h$ *is valid under* $\mathsf{pk_{ch}}$ *or not.*

$r' \leftarrow\!\!\$\ \mathsf{CAdapt}(\mathsf{sk_{ch}}, h, \mu, r, \mu')$: *The algorithm* $\mathsf{CAdapt}$ *takes as input the secret key* $\mathsf{sk_{ch}}$*, a hash* $h$*, a message* $\mu$*, randomness* $r$*, and a new message* $\mu'$*. It then outputs new randomness* $r'$*, such that* $1 \leftarrow \mathsf{CHashCheck}(\mathsf{pk_{ch}}, h, \mu', r')$

**Definition 2 (Correctness of a Chameleon Hash Function).** *A chameleon hash function* $\mathsf{CH}$ *is correct, iff*

$$\begin{gathered}\forall\lambda, \forall(\mathsf{pk_{ch}}, \mathsf{sk_{ch}}) \leftarrow\!\!\$\ \mathsf{CKGen}(1^\lambda), \forall\mu, \mu' \in \mathcal{M}, \\ \forall(h, r) \leftarrow\!\!\$\ \mathsf{CHash}(\mathsf{pk_{ch}}, \mu), \forall r' \leftarrow\!\!\$\ \mathsf{CAdapt}(\mathsf{sk_{ch}}, h, \mu, r, \mu') : \\ \mathsf{CHashCheck}(\mathsf{pk_{ch}}, h, \mu, r) = \mathsf{CHashCheck}(\mathsf{pk_{ch}}, h, \mu', r') = 1\end{gathered}$$

For security we require that the chameleon hash function is collision-resistant, even in the presence of a collision oracle. Furthermore, we require that it hard to distinguish whether a hash randomness pair $(h, r)$ was output by $\mathsf{CH}$ or $\mathsf{CAdapt}$. We model the security games in Figs. 2,3.

**Definition 3.** *We say that a chameleon hash function* $\mathsf{CH}$ *is fully collision-resistant, if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all qpt adversaries* $\mathcal{A}$ *we have that* $\Pr[\mathsf{f\text{-}CR}_{\Pi,\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

$\underline{\mathsf{f\text{-}CR}_{\mathsf{CH},\mathcal{A}}(1^\lambda)}$

$\mathcal{Q} \leftarrow \emptyset$

$(\mathsf{pk}_{\mathsf{ch}}, \mathsf{sk}_{\mathsf{ch}}) \leftarrow\!\$\ \mathsf{CKGen}(1^\lambda)$

$(h, \mu, r, \mu', r') \leftarrow\!\$\ \mathcal{A}^{\mathsf{Adapt}\mathcal{O}}(\mathsf{pk}_{\mathsf{ch}})$

**return** $\big(\mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, h, \mu, r) = 1$

$\quad \wedge\, \mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, h, \mu', r') = 1$

$\quad \wedge\, \mu \neq \mu' \wedge (h, \mu) \notin \mathcal{Q}\big)$

$\underline{\mathsf{Adapt}\mathcal{O}(\mathsf{sk}_{\mathsf{ch}}, \cdot, \cdot, \cdot, \cdot) \text{ with } (h, \mu, r, \mu')}$

**if** $\mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, h, \mu, r) = 0$

$\quad$ **return** $\perp$

$r' \leftarrow\!\$\ \mathsf{CAdapt}(\mathsf{sk}_{\mathsf{ch}}, h, \mu, r, \mu')$

$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(h, \mu), (h, \mu')\}$

**return** $r'$

Fig. 2: f-CR security game.

$\underline{\mathsf{CHIndist}_{\Pi,\mathcal{A},b}(\lambda)}$

$(\mathsf{pk}_{\mathsf{ch}}, \mathsf{sk}_{\mathsf{ch}}) \leftarrow\!\$\ \mathsf{CKGen}(1^\lambda)$

$b' \leftarrow\!\$\ \mathcal{A}^{\mathsf{HashOrAdapt}\mathcal{O}_b, \mathsf{Adapt}\mathcal{O}}(\mathsf{pk}_{\mathsf{ch}})$

**return** $b'$

$\underline{\mathsf{HashOrAdapt}\mathcal{O}_b(\mathsf{sk}_{\mathsf{ch}}, \cdot, \cdot) \text{ with } (\mu, \mu')}$

$(h, r) \leftarrow\!\$\ \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, \mu)$

$(h', r') \leftarrow\!\$\ \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, \mu')$

$r'' \leftarrow\!\$\ \mathsf{CAdapt}(\mathsf{sk}_{\mathsf{ch}}, h', \mu', r', \mu)$

**if** $b = 0$ **return** $(h, r)$

**if** $b = 1$ **return** $(h', r'')$

Fig. 3: CH Indistinguishability game.

There exist other, weaker notions of collision-resistance of a chameleon hash function, called standard, enhanced, and weak collision-resistance. For definitions and comparisons, see [20]. However, we only concern ourselves with the strongest version, which implies the others.

**Definition 4.** *We say that a chameleon hash function* $\mathsf{CH}$ *is indistinguishable, if there exists a negligible function* $\mathsf{negl}(\lambda)$*, such that for all qpt adversaries* $\mathcal{A}$ *we have that* $|\Pr[\mathsf{CHIndist}_{\mathsf{CH},\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{CHIndist}_{\mathsf{CH},\mathcal{A},1}(\lambda) = 1]| \leq \mathsf{negl}(\lambda)$.

### 2.2 Lattices

Next, we cover various relevant definitions and results for lattices.

**Definition 5.** *An* $n$*-dimensional lattice* $\Lambda$ *is a discrete, additive subgroup of* $\mathbb{R}^n$.

**Definition 6.** *In the short integer solution (SIS) problem* $\mathsf{SIS}_{n,m,q,\beta}$ *a qpt adversary* $\mathcal{A}$ *is given a uniformly random* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ *and is asked to compute a* $\mathbf{x} \in \mathbb{Z}^m$ *such that* $\mathbf{A}\mathbf{x} = \mathbf{0}$ *and* $0 < \|\mathbf{x}\| \leq \beta$.

*We define the advantage of a qpt adversary* $\mathcal{A}$ *against* $\mathsf{SIS}_{n,m,q,\beta}$ *as*

$$\mathsf{SIS}_{n,m,q,\beta}(\mathcal{A}) \coloneqq \Pr[\mathbf{A}\mathbf{x} = \mathbf{0} \mod q, 0 < \|\mathbf{x}\| \leq \beta : \mathbf{A} \leftarrow\!\$\ \mathbb{Z}_q^{n \times m}, \mathbf{x} \leftarrow\!\$\ \mathcal{A}(\mathbf{A})].$$

**Definition 7.** *Define the multidimensional Gaussian function with Gaussian parameter* $s$ *and center* $\mathbf{c}$ *as* $\rho_{s,\mathbf{c}}(\mathbf{x}) = \exp(-\|\mathbf{x} - \mathbf{c}\|/s)$*. Then, we define the discrete Gaussian distribution over lattice* $\Lambda$ *with Gaussian parameter* $s$ *and center* $\mathbf{c}$ *as* $D_{\Lambda,s,\mathbf{c}}(\mathbf{x}) = \frac{\rho_{s,\mathbf{c}}(\mathbf{x})}{\rho_{s,\mathbf{c}}(\Lambda)}$.

To construct our chameleon hash, we need a lattice-based trapdoor. We use so-called G-trapdoors.

**Definition 8 ([36]).** *Let $\mathbf{G} \in \mathbb{Z}_q^{n \times w}$ be a so-called gadget matrix, for which SIS is easy. Define a G-trapdoor for a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ to be some matrix $\mathbf{R} \in \mathbb{Z}_q^{m \times w}$ such that $\mathbf{AR} = \mathbf{G} \mod q$.*

We denote the quality of a G-trapdoor by its spectral norm $s_1(\mathbf{R})$. Note that if one has a trapdoor $\mathbf{R}$ for $\mathbf{A}$, then $[\mathbf{R}^t, \mathbf{0}_{w \times k}]^t \in \mathbb{Z}_q^{m+k \times w}$ is a trapdoor for $[\mathbf{A}|\mathbf{B}]$ for any $\mathbf{B} \in \mathbb{Z}_q^{n \times k}$, where $\mathbf{0}_{a \times b}$ is the all-zero matrix of dimension $a \times b$.

With a G-trapdoor, we can invert matrix-vector multiplication of the trapdoored matrix $\mathbf{A}$, where the inverted preimage has a certain (conditioned) Gaussian distribution.

**Theorem 9 ([36]).** *There exists a ppt algorithm $\mathsf{TrapGen}(1^n, 1^m, q)$ that, given any integers $n \geq 1$, $q \geq 2$, and sufficiently large $m = \mathcal{O}(n \log q)$, outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a G-trapdoor $\mathbf{R}$ for $\mathbf{A}$, such that the distribution of $\mathbf{A}$ is negligibly far from uniform. Moreover, there exists a ppt algorithm $\mathsf{PreSample}(\mathbf{A}, \mathbf{R}, \mathbf{u}, s)$ that, given some $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, some G-trapdoor $\mathbf{R} \in \mathbb{Z}_q^{m \times w}$ for $\mathbf{A}$ with $s_1(\mathbf{R}) \in \mathcal{O}(\sqrt{n \log q})$, some $\mathbf{u} \in \mathbb{Z}_q^n$ and large enough $s = \mathcal{O}(\sqrt{n \log q})$, samples from a distribution that is within negligible statistical distance from $D_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}), s \cdot \omega(\sqrt{\log n})}$. Furthermore, the distribution of sampling $\mathbf{x} \leftarrow\!\!\$\ D_{\mathbb{Z}_q^m, s}$, setting $\mathbf{y} = \mathbf{Ax}$, and outputting $(\mathbf{x}, \mathbf{y})$ is statistically indistinguishable from the distribution of choosing $\mathbf{y}' \leftarrow\!\!\$\ \mathbb{Z}_q^n$, computing $\mathbf{x}' \leftarrow\!\!\$\ \mathsf{PreSample}(\mathbf{A}, \mathbf{R}, \mathbf{y}, s)$, and outputting $(\mathbf{x}', \mathbf{y}')$.*

If $\mathbf{R}$ is a trapdoor for $\mathbf{A}$, we sometimes write $\mathsf{PreSample}(\begin{bmatrix}\mathbf{A}\ \mathbf{B}\end{bmatrix}, \mathbf{R}, \mathbf{u}, s)$ instead of $\mathsf{PreSample}(\begin{bmatrix}\mathbf{A}\ \mathbf{B}\end{bmatrix}, \begin{bmatrix}\mathbf{R}^t\ \mathbf{0}^t\end{bmatrix}^t, \mathbf{u}, s)$, since $\mathbf{R}$ can be extended to a trapdoor for $\begin{bmatrix}\mathbf{A}\ \mathbf{B}\end{bmatrix}$ for any $\mathbf{B}$, as noted above.

### 2.3 Trapdoor Commitment Scheme

A trapdoor commitment scheme is a commitment scheme that is (computationally) binding and hiding, but if someone is in possession of a trapdoor to the public parameters, then one can break binding. We model that by introducing an additional ppt algorithm $\mathsf{Equiv}$ that equivocates a commitment $c$ to any message $\mu'$ with the help of the trapdoor.

**Definition 10.** *A trapdoor commitment scheme consists of four ppt algorithms $(\mathsf{TdGen}, \mathsf{Com}, \mathsf{ComCheck}, \mathsf{Equiv})$.*

$(\mathsf{pp}, \mathsf{td}) \leftarrow\!\!\$\ \mathsf{TdGen}(1^\lambda)$: *On input a security parameter $1^\lambda$, the algorithm outputs public parameters $\mathsf{pp}$ and a trapdoor $\mathsf{td}$.*

$(c, d) \leftarrow\!\!\$\ \mathsf{Com}(\mathsf{pp}, \mu)$: *On input some $\mathsf{pp}$ and a message $\mu$, the algorithm outputs a commitment $c$ and an opening value $d$.*

$b \leftarrow \mathsf{ComCheck}(\mathsf{pp}, \mu, c, d)$: *On input some $\mathsf{pp}$, a message $\mu$, a commitment $c$ and an opening value $d$, the algorithm outputs a bit $b$.*

| $\mathsf{Bind}_{\Pi,\mathcal{A}}(\lambda)$ | $\mathsf{DEE}_{\Pi,\mathcal{A},b}(\lambda)$ |
|---|---|
| $(\mathsf{pp},\mathsf{td}) \leftarrow\!\$\ \mathsf{TdGen}(1^\lambda)$ | $(\mathsf{pp},\mathsf{td}) \leftarrow\!\$\ \mathsf{TdGen}(1^\lambda)$ |
| $(c,\mu,d,\mu',d') \leftarrow\!\$\ \mathcal{A}(\mathsf{pp})$ | $\mu \leftarrow\!\$\ \mathcal{A}(\mathsf{pp})$ |
| **if** $\mathsf{ComCheck}(\mathsf{pp},\mu,c,d)=1$ | $(c_0,d_0) \leftarrow\!\$\ \mathsf{Com}(\mathsf{pp},\mu)$ |
| $\quad\wedge\ \mathsf{ComCheck}(\mathsf{pp},\mu',c,d')=1$ | $c_1 \leftarrow\!\$\ \mathsf{Com}(\mathsf{pp},0)$ |
| $\quad\wedge\ \mu\neq\mu'$ | $d_1 \leftarrow\!\$\ \mathsf{Equiv}(\mathsf{pp},\mathsf{td},c_1,\mu)$ |
| $\quad$ **return** 1 | $b' \leftarrow\!\$\ \mathcal{A}(c_b,d_b)$ |
| **return** 0 | **return** $b'$ |

Fig. 4: Trapdoor commitment security games. DEE is distributional equivalence of equivocation.

$d' \leftarrow\!\$\ \mathsf{Equiv}(\mathsf{pp},\mathsf{td},c,\mu')$: *On input some* $\mathsf{pp}$*, a trapdoor* $\mathsf{td}$*, a commitment* $c$ *and a message* $\mu'$*, the algorithm outputs some opening value* $d'$*.*

*A trapdoor commitment scheme is correct, iff*

$$\begin{gathered}\forall\lambda,\forall(\mathsf{pp},\mathsf{td}) \leftarrow\!\$\ \mathsf{TdGen}(1^\lambda),\forall\mu,\mu',\in\mathcal{M},\\ \forall(c,d) \leftarrow\!\$\ \mathsf{Com}(\mathsf{pp},\mu),\forall d' \leftarrow\!\$\ \mathsf{Equiv}(\mathsf{pp},\mathsf{td},c,\mu'),\\ \mathsf{ComCheck}(\mathsf{pp},\mu,c,d)=\mathsf{ComCheck}(\mathsf{pp},\mu',c,d')=1.\end{gathered}$$

We need two security notions for a trapdoor commitment scheme. One is the typical binding, which states that an adversary cannot open a commitment to two different messages. The other security notion is called *distributional equivalence of equivocation*. This requires that, for fixed $\mathsf{pp}$ and $\mu$, the joint distribution of a commitment and an opening value is computationally indistinguishable when either committing to the message or committing to 0 and equivocating to the message. We model these security games in Fig. 4.

**Definition 11.** *We say that a commitment scheme* $\Pi$ *is computationally binding, if there exists a negligible function* $\mathsf{negl}$ *such that for all qpt adversaries* $\mathcal{A}$ *it holds that* $\Pr[\mathsf{Bind}_{\Pi,\mathcal{A}}(\lambda)] \leq \mathsf{negl}(\lambda)$.

**Definition 12.** *We say that a commitment scheme* $\Pi$ *has distributional equivalence of equivocation (DEE), if there exists a negligible function* $\mathsf{negl}$ *such that for all qpt adversaries* $\mathcal{A}$ *it holds that*

$$|\Pr[\mathsf{DEE}_{\Pi,\mathcal{A},0}(\lambda)=1]-\Pr[\mathsf{DEE}_{\Pi,\mathcal{A},1}(\lambda)=1]| \leq \mathsf{negl}(\lambda).$$

It is possible to adapt known lattice-based commitment schemes such as Ajtai [1] or BDLOP [5] to be trapdoor commitment schemes by replacing their uniform generation of matrices by $\mathsf{TrapGen}$. Then, one uses $\mathsf{PreSample}$ to construct $\mathsf{Equiv}$, and due to the properties mentioned in Theorem 9 we can show DEE.

Note that we did not define the typical notion of a hiding commitment scheme as we do not need it explicitly for our construction. However, hiding is implied

by DEE, since in the DEE security game $c_1$ is generated by committing to 0. If the commitment scheme was not hiding, the adversary could get information about $b$ by $c_b$ alone. Note that binding trapdoor commitment schemes with DEE are strongly related to collision-resistant preimage-sampleable functions [24].

### 2.4 Linking Indistinguishable Tag

To later construct the VRS we need so-called *linking indistinguishable tags* (LIT) [7, 9]. This is a secret key tagging scheme like a MAC, except that there exists an additional linking algorithm. With this, it is possible to detect if the same person tagged the same message twice. Additionally, there exists a function $f$ that computes a public key from a secret key. While this is not used for the LIT itself, it is useful when using LITs to construct other schemes. For security we require that it is hard to trick the linking algorithm as well as unforgeability similar to a MAC, but also that apart from linking, it is hard to decide which person created a tag. Also it should be hard to compute a secret key from a public key and tags.

**Definition 13.** *A linking indistinguishable tag scheme consists of a function $f$ and the following ppt algorithms:*

$sk \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$**:** *On input a security parameter $1^\lambda$, it outputs a secret $sk$.*
$t \leftarrow\!\!\$\ \mathsf{Tag}(sk, \mu)$**:** *On input a secret key $sk$ and a message $\mu$, it outputs a tag $t$.*
$b \leftarrow \mathsf{Vrfy}(sk, \mu, t)$**:** *On input a secret key $sk$, a message $\mu$ and a tag $t$, it outputs a bit $b$.*
$b \leftarrow \mathsf{Link}(\mu, t_0, t_1)$**:** *On input a message $\mu$ and two tags $t_0, t_1$, it outputs a bit $b$.*

*We require that a LIT is correct. This is the case if*

$$\forall \lambda, \forall sk \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda), \forall \mu, \forall t_0, t_1 \leftarrow\!\!\$\ \mathsf{Tag}(sk, \mu) : \\ \mathsf{Vrfy}(sk, \mu, t_0) = \mathsf{Link}(\mu, t_0, t_1) = 1.$$

For the security model, we use the one from [9], with one change improving the security. We use the same games for tag-indistinguishability, non-invertability, linkability, and unforgeability. However, as can be seen in Fig. 5, when defining the oracles, we no longer return a previously computed tag if a message has been previously queried. Instead, the adversary gets a fresh tag every time it queries the oracle.

**Definition 14.** *A LIT $\Pi$ has tag-indistinguishability, if there exists a negligible function* $\mathsf{negl}$ *such that for all ppt adversaries $\mathcal{A}$ it holds that*

$$\mathsf{Adv}^{LITAnon}_{\Pi,\mathcal{A}}(\lambda) := \left|\Pr[\mathsf{Anon}^{\mathsf{LIT}}_{\Pi,\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Anon}^{\mathsf{LIT}}_{\Pi,\mathcal{A},1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda).$$

**Definition 15.** *A LIT $\Pi$ has linkability if there exists a negligible function* $\mathsf{negl}$ *such that for all ppt adversaries $\mathcal{A}$ it holds that*

$$\Pr[\mathsf{Linkable}^{\mathsf{LIT}}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

$\underline{\mathsf{Anon}^{\mathsf{LIT}}_{\Pi,\mathcal{A},b}(\lambda)}$

$\mathcal{Q} \leftarrow \emptyset$
$sk_0, sk_1 \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$
$pk_i \leftarrow f(sk_i), i \in \{0,1\}$
$\mu^* \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Tag}\mathcal{O}}(pk_0, pk_1)$
$t^* \leftarrow\!\!\$\ \mathsf{Tag}(sk_b, \mu^*)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Tag}\mathcal{O}}(t^*)$
**if** $(\mu^*, \cdot) \in \mathcal{Q}$
  **return** 0
**return** $b'$

$\underline{\mathsf{Linkable}^{\mathsf{LIT}}_{\Pi,\mathcal{A}}(\lambda)}$

$(sk_0, sk_1, \mu, t_0, t_1) \leftarrow\!\!\$\ \mathcal{A}(1^\lambda)$
**if** $f(sk_0) \neq f(sk_1)$
  **return** 0
**if** $\exists i \in \{0,1\} :\ \mathsf{Vrfy}(sk_i, \mu, t_i) = 0$
  **return** 0
**if** $\mathsf{Link}(\mu, t_0, t_1) = 0$
  **return** 1
**return** 0

$\underline{\mathsf{Tag}\mathcal{O}(\{sk_j\}_j, \cdot, \cdot) \text{ with } (i, \mu)}$

$t \leftarrow\!\!\$\ \mathsf{Tag}(sk_i, \mu)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mu, t)\}$
**return** $t$

$\underline{\mathsf{Invert}^{\mathsf{LIT}}_{\Pi,\mathcal{A}}(\lambda)}$

$sk \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$
$pk_0 \leftarrow f(sk_0)$
$sk' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Tag}\mathcal{O}}(pk_0)$
**if** $pk = f(sk')$
  **return** 1
**return** 0

$\underline{\mathsf{Forge}^{\mathsf{LIT}}_{\Pi,\mathcal{A}}(\lambda)}$

$\mathcal{Q} \leftarrow \emptyset$
$sk_0 \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda), pk_0 \leftarrow f(sk)$
$(sk^*, \mu, t^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Tag}\mathcal{O}}(pk_0)$
**if** $\mathsf{Vrfy}(sk^*, \mu, t^*) = 0$
  **return** 0.
**if** $\exists\ (\mu, t) \in \mathcal{Q} : \mathsf{Link}(\mu, t, t^*) = 1$
  **return** 1
**return** 0

Fig. 5: LIT security games.

**Definition 16.** *A LIT $\Pi$ is unforgeable, if there exists a negligible function* $\mathsf{negl}$ *such that for all ppt adversaries $\mathcal{A}$ it holds that*

$$\Pr[\mathsf{Forge}^{\mathsf{LIT}}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

**Definition 17.** *A LIT $\Pi$ has non-invertability, if there exists a negligible function* $\mathsf{negl}$ *such that for all ppt adversaries $\mathcal{A}$ it holds that*

$$\Pr[\mathsf{Invert}^{\mathsf{LIT}}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

Due to the aforementioned change in security model, the construction of a LIT from [9] is not secure in the new model. This is because if this construction tags the same message twice, the adversary receives two *learning with errors* (LWE) samples with the same $\mathbf{A}$. To remedy this, we propose a construction similar to that of [9], that instead of relying on LWE uses *learning with rounding* (LWR) [4]. Due to this, tagging becomes deterministic and linking becomes an equality

check. Thus, naturally there is only one tag for each message under a secret key. Therefore, with the LWR-based LIT the security model of [9] and our new one look the same for an adversary, which is why we can then do the same reductions as for the LWE-based LIT of [9] to show that the deterministic LIT is secure if LWR is hard. Thus, the following lemma follows directly from [9].

**Lemma 18.** *There exists a LIT that has tag-indistinguishability, linkability, unforgeability, and invertability if decisional LWR and search LWR are hard.*

For the formal proof, see Section F.1.

### 2.5 Non-Interactive Zero-Knowledge Proof

We model non-interactive zero-knowledge proof systems in the random oracle model as in [9].

**Definition 19 (NIZK).** *A non-interactive proof system (NIZK) for a relation $\mathfrak{R}$ in the random oracle model is defined as a triple $\Pi_{\mathsf{NIZK}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ of ppt algorithms:*

$\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$**:** *On input $1^\lambda$ the setup algorithm outputs a common reference string* crs*.*

$\pi \leftarrow\!\!\$\ \mathsf{P}^{\mathcal{RO}}(\mathsf{crs}, x, w, \mu)$**:** *On input a common reference string* crs*, an instance $x$, witness $w$, and a message $\mu$, and given oracle access to the random oracle $\mathcal{RO}$, the prover outputs a proof $\pi$.*

$b \leftarrow \mathsf{V}^{\mathcal{RO}}(\mathsf{crs}, x, \mu, \pi)$**:** *On input a common reference string* crs*, a statement $x$, a message $\mu$, and a proof $\pi$, and given oracle access to the random oracle $\mathcal{RO}$, the verifier outputs a bit $b$.*

*To simplify notation, we sometimes omit the random oracle $\mathcal{RO}$, but assume implicitly that the prover and verifier have access to it. We say that the NIZK is* correct*, if for all $(x, w) \in \mathfrak{R}$ and $m \in \{0,1\}^*$, we have that*

$$\Pr[\mathsf{V}(\mathsf{crs}, x, m, \mathsf{P}(\mathsf{crs}, x, w, m)) : \mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)] = 1.$$

The message in the above definition is not necessary for a NIZK itself, but since the NIZK we use uses the Fiat-Shamir heuristic, we want to be able make the proof dependent on a message. For a relation $\mathfrak{R}$, we define $L_{\mathfrak{R}} = \{x \mid \exists w : (x, w) \in \mathfrak{R}\}$ as the language of $\mathfrak{R}$. We define a shorthand notation to quickly show what relation we want to prove.

**Definition 20.** *We denote the generation of a proof $\pi \leftarrow\!\!\$\ \mathsf{P}(\mathsf{crs}, x, w, \mu)$ by*

$$\pi \leftarrow\!\!\$\ \mathsf{NIZK}\{x; w; \mathfrak{R}(x, w)\}(\mu),$$

*where* P *is from a non-interactive proof system $\Pi_{\mathsf{NIZK}}$ for the relation $\mathfrak{R}$.*

For security, we use the standard notions of zero-knowledge and straight-line extractability.

**Definition 21 (Zero-Knowledge).** *A NIZK $\Pi$ is* zero-knowledge *if there exists a simulator* $\mathsf{Sim}$ *consisting of three ppt algorithms* $\mathsf{Sim} = (\mathsf{Sim.Setup}, \mathsf{Sim}.\mathcal{RO}, \mathsf{Sim.Sim})$ *such that for all ppt* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}$ *such that,*

$$\left| \begin{array}{l} \Pr[\mathcal{A}^{\mathsf{P}\mathcal{O},\mathcal{RO}}(1^\lambda, \mathsf{crs}) = 1 : \mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)] \\ -\ \Pr[\mathcal{A}^{\mathsf{Sim.Sim},\mathsf{Sim}.\mathcal{RO}}(1^\lambda, \mathsf{crs}) = 1 : \mathsf{crs} \leftarrow\!\!\$\ \mathsf{Sim.Setup}(1^\lambda)] \end{array} \right| \leq \mathsf{negl}(\lambda)$$

*where* $\mathcal{RO}$ *denotes a random oracle and* $\mathsf{P}\mathcal{O}$*, queried on input* $(x, w, \mu)$*, returns* $\mathsf{P}(\mathsf{crs}, x, w, \mu)$*. The oracle* $\mathsf{Sim}(x, w, \mu)$ *checks if* $(x, w) \in \mathfrak{R}$ *and if so, returns* $\mathsf{Sim.Sim}(x, \mu)$*. We assume that* $\mathsf{Sim}$ *is stateful, i.e., it implicitly keeps state between invocations of* $\mathsf{Sim.Setup}$*,* $\mathsf{Sim}.\mathcal{RO}$*, and* $\mathsf{Sim.Sim}$*.*

**Definition 22 (Straight-line extractability).** *Let* $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *be a NIZK. We say that* $\Pi$ *is a* straight-line extractable proof of knowledge *if there are ppt algorithms* $\mathsf{Ext}_0, \mathsf{Ext}_1$ *such that for all ppt* $\mathcal{A}_0, \mathcal{A}_1$*, there exist negligible functions* $\mathsf{negl}_0, \mathsf{negl}_1$ *such that*

$$\left| \begin{array}{l} \Pr[\mathcal{A}_0(1^\lambda, \mathsf{crs}) = 1 : \mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)] \\ -\ \Pr[\mathcal{A}_0(1^\lambda, \mathsf{crs}) = 1 : (\mathsf{crs}, \mathsf{td}) \leftarrow\!\!\$\ \mathsf{Ext}_0(1^\lambda)] \end{array} \right| \leq \mathsf{negl}_0(\lambda)$$

*and*

$$\Pr\left[ \begin{array}{l} \mathsf{V}^{\mathcal{RO}}(\mathsf{crs}, x, m, \pi) = 1 \\ \wedge\ (x, w) \notin \mathfrak{R} \end{array} : \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow\!\!\$\ \mathsf{Ext}_0(1^\lambda), \\ (x, m, \pi) \leftarrow\!\!\$\ \mathcal{A}_1(1^\lambda, \mathsf{crs}), \\ w \leftarrow \mathsf{Ext}_1(\mathsf{td}, x, m, \pi) \end{array} \right] \leq \mathsf{negl}_1(\lambda)$$

*In the random oracle model,* $\mathsf{Ext}_1$ *gets the list of random oracle queries that* $\mathcal{A}$ *made as additional input.*

### 2.6 Verifiable Ring Signature

A verifiable ring signature (VRS) is a standard ring signature with an additional functionality. At any point after creating a signature, a signer can output a proof showing that in fact it created the signature. On the other hand, people who did not sign the signature, but are part of the ring, can show that they did *not* create the signature. We formalize this by adding two ppt algorithms $\mathsf{Prove}$ and $\mathsf{Judge}$ to the standard ring signature model.

**Definition 23.** *A verifiable ring signature consists of five ppt algorithms* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Link})$*.*

$\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$: *On input a security parameter* $1^\lambda$*, the setup algorithm outputs public parameters* $\mathsf{pp}$*.*

$(sk, pk) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{pp})$: *On input public parameters* $\mathsf{pp}$*, the key generation algorithm outputs a secret, public key pair* $(sk, pk)$*.*

$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk, R, \mu)$: *On input a secret key* $sk$*, a ring* $R = \{pk_i\}_i$ *and a message* $\mu$*, the signing algorithm outputs a signature* $\sigma$*.*

$b \leftarrow \mathsf{Vrfy}(R, \mu, \sigma)$: *On input a ring* $R$*, a message* $\mu$ *and a signature* $\sigma$*, the verifying algorithm outputs a bit* $b$*.*

$\pi \leftarrow\!\!\$\ \mathsf{Prove}(sk, R, \mu, \sigma)$: *On input a secret key $sk$, a ring $R$, a message $\mu$, and a signature $\sigma$, the prove algorithm outputs a proof $\pi$.*

$b \leftarrow \mathsf{Judge}(pk, R, \mu, \sigma, \pi)$: *On input a public key $pk$, a ring $R$, a message $\mu$, a signature $\sigma$, and a proof $\pi$, the judging algorithm outputs a bit $b$.*

For security, we require strong unforgeability and anonymity of a ring signature, but now the adversary also has an oracle generating proofs with the Prove algorithm. Additionally we require *accountability*, which means that it is hard for an adversary to create a signature, where Judge thinks it did not create it. Furthermore we require *non-seizability*, which means that is hard for an adversary to create a signature, where Judge attributes the signature to an honest user. To model this, we use the security model of [15]. The definition of the games can be found in Fig. 6.

**Definition 24.** *We say that a VRS $\Pi$ is strongly unforgeable, if there exists a negligible function* negl *such that for all qpt adversaries $\mathcal{A}$ and all $\ell \in \mathsf{poly}(\lambda)$ it holds that*

$$\Pr[\mathsf{Forge}^{\ell}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

**Definition 25.** *We say that a VRS $\Pi$ is anonymous, if there exists a negligible function* negl *such that for all qpt adversaries $\mathcal{A}$ it holds that*

$$\left|\Pr[\mathsf{Anon}^{0}_{\Pi,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Anon}^{1}_{\Pi,\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda).$$

**Definition 26.** *We say that a VRS $\Pi$ is strongly accountable, if there exists a negligible function* negl *such that for all qpt adversaries $\mathcal{A}$ and all $\ell \in \mathsf{poly}(\lambda)$ it holds that*

$$\Pr[\mathsf{Acc}^{\ell}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

**Definition 27.** *We say that a VRS $\Pi$ is strongly non-seizable, if there exists a negligible function* negl *such that for all qpt adversaries $\mathcal{A}$ it holds that*

$$\Pr[\mathsf{Seiz}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

## 3 New Lattice Constructions

### 3.1 A Fully Collision-Resistant Chameleon Hash Function

We now want to construct our new chameleon hash to later use it when instantiating SSS constructions. Since the security notion that [31] achieves is close to our target security, full collision-resistance, we take their construction as a starting point. The idea of their random oracle model construction is to first generate a trapdoored matrix $\mathbf{A}$ in the setup. To hash a message $\mu$, they query the random oracle on $\mu$ and some other values to get another matrix $\mathbf{A}_h$ and choose a small Gaussian value $\mathbf{e}$. Then, they use the well known Ajtai hash function to compute $[\mathbf{A}|\mathbf{A}_h]\mathbf{e} = \mathbf{h}$, which is their hash value. To adapt a message, they use the trapdoor to compute some short $\mathbf{e}'$ such that $[\mathbf{A}|\mathbf{A}'_h]\mathbf{e}' = \mathbf{h}$ for some different

$\mathsf{Sign}\mathcal{O}(\{sk_i\}_i, \cdot, \cdot, \cdot)$ with $(j, R, \mu)$

$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk_j, R, \mu)$
$\mathcal{Q}_{\mathsf{Sign}} \leftarrow \mathcal{Q}_{\mathsf{Sign}} \cup \{\mu, \sigma\}$
**return** $\sigma$

$\mathsf{Prove}\mathcal{O}(\{sk_i\}_i, \cdot, \cdot, \cdot, \cdot)$ with $(j, R, \mu, \sigma)$

**if** $pk_i \in R \wedge (\mu, \sigma) \notin \mathcal{Q}_{\mathsf{LoR}}$
  **return** $\mathsf{Prove}(sk_j, R, \mu, \sigma)$
**else**
  **return** $\perp$

$\mathsf{LoRSign}\mathcal{O}_b(sk_0, sk_1, \cdot, \cdot)$ with $(R, \mu)$

**if** $\{pk_0, pk_1\} \subseteq R$
  $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk_b, R, \mu)$
  $\mathcal{Q}_{\mathsf{LoR}} \leftarrow \mathcal{Q}_{\mathsf{LoR}} \cup \{\mu, \sigma\}$
  **return** $\sigma$
**else**
  **return** $\perp$

$\mathsf{Forge}^{\ell}_{\Pi,\mathcal{A}}(\lambda)$

$\mathcal{Q}_{\mathsf{Sign}} \leftarrow \emptyset, \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
$\forall i \in [\ell], (sk_i, pk_i) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{pp})$
$(R^*, \mu^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Sign}\mathcal{O},\mathsf{Prove}\mathcal{O}}(\{pk_i\}_{i\in[\ell]})$
$b_0 \leftarrow \mathsf{Vrfy}(R^*, \mu^*, \sigma^*)$
$b_1 \leftarrow R^* \subseteq \{pk_i\}_{i\in[\ell]}$
$b_2 \leftarrow (\mu^*, \sigma^*) \notin \mathcal{Q}_{\mathsf{Sign}}$
**return** $b_0 \wedge b_1 \wedge b_2$

$\mathsf{Anon}^{b}_{\Pi,\mathcal{A}}(\lambda)$

$\mathcal{Q}_{\mathsf{LoR}} \leftarrow \emptyset, \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
$(sk_0, pk_0) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{pp})$
$(sk_1, pk_1) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{pp})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Sign}\mathcal{O},\mathsf{Prove}\mathcal{O},\mathsf{LoRSign}\mathcal{O}}(pk_0, pk_1)$
**return** $b'$

$\mathsf{Acc}^{\ell}_{\Pi,\mathcal{A}}(\lambda)$

$\mathcal{Q}_{\mathsf{Sign}} \leftarrow \emptyset, \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
$\forall i \in [\ell], (sk_i, pk_i) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{pp})$
$(pk^*, R^*, \mu^*, \sigma^*, \pi^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Sign}\mathcal{O},\mathsf{Prove}\mathcal{O}}(\{pk_i\}_{i\in[\ell]})$
$b_0 \leftarrow \mathsf{Vrfy}(R^*, \mu^*, \sigma^*)$
$b_1 \leftarrow (\mathsf{Judge}(pk^*, R^*, \mu^*, \sigma^*, \pi^*) = 0)$
$b_2 \leftarrow R^* \subseteq \{pk_i\}_{i\in[\ell]} \cup \{pk^*\}$
$b_3 \leftarrow (\mu^*, \sigma^*) \notin \mathcal{Q}_{\mathsf{Sign}}$
**return** $b_0 \wedge b_1 \wedge b_2 \wedge b_3$

$\mathsf{Seiz}_{\Pi,\mathcal{A}}(\lambda)$

$\mathcal{Q}_{\mathsf{Sign}} \leftarrow \emptyset, \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$
$(sk_0, pk_0) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{pp})$
$(R^*, \mu^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Sign}\mathcal{O},\mathsf{Prove}\mathcal{O}}(pk_0)$
$\pi^* \leftarrow\!\!\$\ \mathsf{Prove}(sk_0, R^*, \mu^*, \sigma^*)$
$b_0 \leftarrow \mathsf{Vrfy}(R^*, \mu^*, \sigma^*)$
$b_1 \leftarrow (\mathsf{Judge}(pk_0, R^*, \mu^*, \sigma^*, \pi^*) \neq 0)$
$b_2 \leftarrow (\mu^*, \sigma^*) \notin \mathcal{Q}_{\mathsf{Sign}}$
**return** $b_0 \wedge b_1 \wedge b_2$

Fig. 6: VRS security games and oracles.

$\mathbf{A}'_h$ also output by the random oracle. The issue with this idea is that to show security they need to assume having a unique tag $\tau$ each time a new $\mathbf{h}$ is created, which they use as additional input to the random oracle. While this is no problem in their application of redactable blockchains, the constructions of SSS that we look at require the chameleon hash to be tag-free. If we simply removed the tag from their construction, their construction would not be fully collision-resistant. An adversary could query the adapt oracle for a collision $[\mathbf{A}|\mathbf{A}_h]\mathbf{e} = [\mathbf{A}|\mathbf{A}'_h]\mathbf{e}'$ and return $(2\mathbf{e}, 2\mathbf{e}')$ as a new collision which wins the security game.

The problem with this approach is that $\mathbf{e}$ can be freely chosen by the adversary without any restrictions. Therefore, the idea for our construction is to use the construction of [31] without tags and to additionally bind $\mathbf{e}$ with a commitment scheme to prohibit this attack. In fact, a commitment scheme is not sufficient, as we then would not be able to implement the $\mathsf{CAdapt}$ algorithm correctly. Instead, we use a trapdoor commitment scheme and its DEE property.

We can now construct our chameleon hash function. For this, let $n, q > 1$, $m = \mathcal{O}(n \log q)$, $s = \mathcal{O}(\sqrt{n \log q})$ large enough, and $\beta = s \cdot \sqrt{2m}$. Let $\Pi_{\mathsf{Com}} = (\mathsf{TdGen}, \mathsf{Com}, \mathsf{ComCheck}, \mathsf{Equiv})$ be a trapdoor commitment scheme. Let the random oracle $\mathcal{RO} : \{0,1\}^* \to \mathbb{Z}_q^{n \times m}$. We then construct our chameleon hash function as seen in Fig. 7.

Note that we require $\mathbf{e}_2 \neq \mathbf{0}$ for security to hold (which we show in Appendix F.2).

**Theorem 28.** *If the commitment scheme has DEE, then the chameleon hash function construction given in Fig. 7 has indistinguishability.*

*Proof.* Let $D_{\mathsf{CHash}}$ be the distribution of $(\mathbf{h}, r = (z, \mathbf{e}, c, d) \leftarrow\!\$\ \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, \mu))$ and let $D_{\mathsf{CAdapt}}$ be the distribution of $(\mathbf{h}', r'' = (z'', \mathbf{e}'', c'', d''))$, where $(\mathbf{h}', r') \leftarrow\!\$\ \mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, \mu'), r'' \leftarrow\!\$\ \mathsf{CAdapt}(\mathsf{sk}_{\mathsf{ch}}, \mathbf{h}', \mu', r', \mu)$ for some $\mu, \mu'$. Then we can easily see the marginal distributions of $\mathbf{h}$ and $\mathbf{h}'$ are statistically close to uniformly random by Theorem 56. Furthermore, $z, z''$ are uniform by definition. For $\mathbf{e}, \mathbf{e}''$ we know that their distribution is statistically indistinguishable due to Theorem 9. Finally, by the DEE property of the commitment scheme we know that $(c, d), (c'', d'')$ are computationally indistinguishable. □

**Lemma 29.** *The construction given in Fig. 7 does* not *have uniqueness (cf. Definition 54).*

This can be shown by the following attack. Since the adversary in the uniqueness game can choose the public key, it can generate a public key honestly together with a trapdoor. Then, it can just hash some message $\mu$ to $(h, r)$ and use the $\mathsf{CAdapt}$ algorithm with input $(\mathsf{sk}_{\mathsf{ch}}, h, \mu, r, \mu)$ to get some new $r' = (z', \mathbf{e}', c', d')$ for the same message $\mu$. While it is possible to derandomize $z', c', d'$, such that $z = z', c = c', d = d'$ (for the same $\mu$ and $\mathbf{e}$) with the help of a PRF, by design, $\mathbf{e}' \neq \mathbf{e}$ with overwhelming probability, since the output distribution of $\mathsf{PreSample}$ is a (conditioned) discrete Gaussian. We expect it is necessary that in order to construct a chameleon hash function with uniqueness, one needs to use another building block than $\mathsf{PreSample}$.

$\mathsf{CKGen}(1^\lambda)$

$(\mathbf{A}, \mathsf{td}) \leftarrow\!\!\$\ \mathsf{TrapGen}(n, m, q, s)$
$(\mathsf{pp}, \mathsf{td}_{\mathsf{Com}}) \leftarrow\!\!\$\ \mathsf{TdGen}(1^\lambda)$
$\mathsf{pk}_{\mathsf{ch}} \leftarrow (\mathbf{A}, \mathsf{pp})$
$\mathsf{sk}_{\mathsf{ch}} \leftarrow (\mathsf{td}, \mathsf{td}_{\mathsf{Com}})$
**return** $(\mathsf{pk}_{\mathsf{ch}}, \mathsf{sk}_{\mathsf{ch}})$

$\mathsf{CHash}(\mathsf{pk}_{\mathsf{ch}}, \mu)$

$z \leftarrow\!\!\$\ \{0,1\}^\lambda$
$\mathbf{e} \leftarrow\!\!\$\ D_{\mathbb{Z}^{2m}, s}$
$(c, d) \leftarrow\!\!\$\ \mathsf{Com}(\mathsf{pp}, \mathbf{e})$
$\mathbf{B} \leftarrow \mathcal{RO}(\mu, z, c)$
$\mathbf{h} \leftarrow \left[\mathbf{A}\ \mathbf{B}\right] \mathbf{e}$
$h \leftarrow \mathbf{h}, r \leftarrow (z, \mathbf{e}, c, d)$
**return** $(h, r)$

$\mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, h, \mu, r)$

Parse $(z, \mathbf{e} = (\mathbf{e}_1^t, \mathbf{e}_2^t)^t, c, d) \leftarrow r$
  with $\mathbf{e}_2 \in \mathbb{Z}_q^m$, and $\mathbf{h} \leftarrow h$
$\mathbf{B} \leftarrow \mathcal{RO}(\mu, z, c)$
**if** $\mathbf{h} = \left[\mathbf{A}\ \mathbf{B}\right] \mathbf{e}$
  $\wedge\ \mathsf{ComCheck}(\mathsf{pp}, \mathbf{e}, c, d) = 1$
  $\wedge\ \|\mathbf{e}\| \leq \beta \wedge \mathbf{e}_2 \neq \mathbf{0}$
  **return** 1
**return** 0

$\mathsf{CAdapt}(\mathsf{sk}_{\mathsf{ch}}, h, \mu, r, \mu')$

$\mathbf{h} \leftarrow h$
**if** $\mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, h, \mu, r) = 0$
  **return** $\perp$
$z' \leftarrow\!\!\$\ \{0,1\}^\lambda$
$c' \leftarrow\!\!\$\ \mathsf{Com}(\mathsf{pp}, 0)$
$\mathbf{B} \leftarrow \mathcal{RO}(\mu', z', c')$
$\mathbf{e}' \leftarrow\!\!\$\ \mathsf{PreSample}(\left[\mathbf{A}\ \mathbf{B}\right], \mathsf{td}, \mathbf{h}, s)$
$d' \leftarrow\!\!\$\ \mathsf{Equiv}(\mathsf{pp}, \mathsf{td}_{\mathsf{Com}}, c', \mathbf{e}')$
$r' \leftarrow (z', \mathbf{e}', c', d')$
**return** $r'$

Fig. 7: Lattice-based fully-collision-resistant chameleon hash function.

**Theorem 30.** *If* $\mathsf{SIS}_{n,m,q,\beta'}$ *is hard, where* $\beta' = 2s' \cdot (\sqrt{n} + \sqrt{m} + t + 1) \cdot s\sqrt{2m}$ *with* $t \geq 0$ *and* $s' = \omega(\sqrt{\log m})$*, and if* $\Pi$ *is a computationally binding trapdoor commitment scheme that has DEE, the construction* $\mathsf{CH}$ *given in Fig. 7 is fully collision-resistant in the random oracle model.*

The idea of the proof is as follows. First, we take an adversary $\mathcal{A}$ against the full collision-resistance of the construction. Then, we define an alternative security game for $\mathcal{A}$ to play in, in which we do not generate a trapdoor for $\mathbf{A}$ and forget the trapdoor for the commitment scheme. Instead, when answering a $\mathsf{CAdapt}$ query, we use the random oracle to program a trapdoor into the $\mathbf{B}$ we generate during it. However, when we answer a random oracle query, we do not program a trapdoor into $\mathbf{B}$. Then, we want to construct an adversary $\mathcal{B}$ against SIS that simulates $\mathcal{A}$ in the alternative security game. If $\mathcal{A}$ wins this game by outputting a valid collision $(h, \mu, r, \mu', r')$, there are multiple cases that can happen. First, for $(h, \mu, r = (z, \mathbf{e}, c, d))$ we look at $\mathbf{B} \leftarrow \mathcal{RO}(\mu, z, c)$. If $\mathbf{B}$ was generated without a trapdoor, we are fine and can use it to extract an SIS solution. However, we need a second part to do that. In the case that $(h, \mu') \notin \mathcal{Q}$, we hope that $\mathbf{B}' \leftarrow \mathcal{RO}(\mu', z', c')$, where $r' = (z', \mathbf{e}', c', d')$, was

generated without a trapdoor as well. In the other case, i.e., $(h, \mu') \in \mathcal{Q}$, we can argue that there must exist a candidate $(h, \hat{\mu}, \hat{r} = (\hat{z}, \hat{\mathbf{e}}, \hat{c}, \hat{d}))$. For this candidate, we hope that $\hat{\mathbf{B}} \leftarrow \mathcal{RO}(\hat{\mu}, \hat{z}, \hat{c})$ was generated without a trapdoor as well. Finally, if $\mathbf{B}$ was generated without trapdoor and either $\mathbf{B}'$ or $\hat{\mathbf{B}}$ was generated without trapdoor (in the respective case), we can extract an SIS solution. We will later formally define this as the event $\mathsf{Free}$. If, on the other hand, we have $\neg\mathsf{Free}$, we can instead break the binding of the commitment scheme. This is because with the commitment scheme we bind $\mathbf{e}$ to some $\mathbf{B}$ and therefore to some $h$. If $\neg\mathsf{Free}$ and thus one of the $\mathbf{B}$ has a trapdoor in it, the adversary used some $\mathbf{B}$ with two different $\mathbf{e}, \mathbf{e}'$. A full, formal version of this proof can be found in Appendix F.2.

Since we argued before that a lattice-based trapdoor commitment scheme with the required properties exist, we get the following corollary.

**Corollary 31.** *There exists a construction of a chameleon hash function that is f-CR secure, if SIS is hard.*

The fully-collision-resistant chameleon hash function based on lattice assumptions we constructed here, will be applied to instantiate various SSS constructions in Section 5.

### 3.2 A Generic Verifiable Ring Signature Construction, Instantiated with Lattices

Let $\mathsf{LIT} = (\mathsf{LIT.KGen}, \mathsf{Tag}, \mathsf{LIT.Vrfy}, \mathsf{Link}, f)$ $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{P}, \mathsf{V})$ respectively be a LIT and NIZK for the relations required in our construction of a VRS, which can be found in Fig. 8.

**Theorem 32.** *If the LIT has non-invertability and if the NIZK is straight-line extractable and zero-knowledge, then the VRS construction has strong unforgeability.*

*Proof.* Let $\ell \in \mathsf{poly}(\lambda)$ and let $\mathcal{A}$ be a *qpt* adversary against the strong unforgeability of the VRS construction. We construct an adversary $\mathcal{B}$ against the non-invertability of the LIT. On input a $pk$, $\mathcal{B}$ first samples $\ell$ keys $sk_i \leftarrow\!\!\$\ \mathsf{LIT.KGen}(1^\lambda)$ and sets $pk_i \leftarrow f(sk_i)$. Then, it guesses a $k \leftarrow\!\!\$\ [\ell]$ and replaces the $k$th public key $pk_k \leftarrow pk$. $\mathcal{B}$ then simulates $\mathcal{A}$ as in the strong unforgeability game, except for the following two changes.

- When $\mathcal{A}$ makes a query $(i, R, \mu)$ to its $\mathsf{Sign}$ oracle with $i = k$, $\mathcal{B}$ generates the tag with $t_{\mathsf{Sign}} \leftarrow\!\!\$\ \mathsf{Tag}\mathcal{O}((R, \mu, r_{\mathsf{Sign}}))$, where $r_{\mathsf{Sign}} \leftarrow\!\!\$\ \{0,1\}^\lambda$ as in the construction. The proof $\pi$ is generated with the simulator of the NIZK.
- When $\mathcal{A}$ makes a query $(i, R, \mu, \sigma)$ to its $\mathsf{Prove}\mathcal{O}$ oracle with $i = k$, and if $pk \in R$ and $\mathsf{Vrfy}(R, \mu, \sigma) = 1$, and where $\sigma = (r_{\mathsf{Sign}}, t_{\mathsf{Sign}}, \pi_{\mathsf{Sign}})$, then $\mathcal{B}$ generates the tag with $t_{\mathsf{Prove}} \leftarrow\!\!\$\ \mathsf{Tag}\mathcal{O}((R, \mu, r_{\mathsf{Sign}}))$. The proof is generated with the simulator of the NIZK.

$\mathsf{Setup}(1^\lambda)$

$\mathsf{pp} \leftarrow\!\!\$\ \mathsf{NIZK.Setup}(1^\lambda)$
**return** $\mathsf{pp}$

$\mathsf{KGen}(\mathsf{pp})$

$sk \leftarrow\!\!\$\ \mathsf{LIT.KGen}(1^\lambda)$
$pk \leftarrow f(sk)$
**return** $(sk, pk)$

$\mathsf{Sign}(sk, R, \mu)$

**if** $f(sk) \notin R$
  **return** $\perp$
$r_{\mathsf{Sign}} \leftarrow\!\!\$\ \{0,1\}^\lambda$
$t_{\mathsf{Sign}} \leftarrow\!\!\$\ \mathsf{Tag}(sk, (R, \mu, r_{\mathsf{Sign}}))$
$\pi_{\mathsf{Sign}} \leftarrow\!\!\$\ \mathsf{NIZK}\{(R, \mu, r_{\mathsf{Sign}}, t_{\mathsf{Sign}});$
  $(sk, pk);$
  $f(sk) = pk, pk \in R,$
  $\mathsf{LIT.Vrfy}(sk, (R, \mu, r_{\mathsf{Sign}}), t_{\mathsf{Sign}}) = 1\}$
$\sigma \leftarrow (r_{\mathsf{Sign}}, t_{\mathsf{Sign}}, \pi_{\mathsf{Sign}})$
**return** $\sigma$

$\mathsf{Vrfy}(R, \mu, \sigma)$

**return** $\mathsf{V}(\mathsf{crs}, (R, \mu, r_{\mathsf{Sign}}, t_{\mathsf{Sign}}), \pi_{\mathsf{Sign}}) = 1$

$\mathsf{Prove}(sk, R, \mu, \sigma)$

**if** $f(sk) \notin R \vee \mathsf{Vrfy}(R, \mu, \sigma) = 0$
  **return** $\perp$
$t_{\mathsf{Prove}} \leftarrow\!\!\$\ \mathsf{Tag}(sk, (R, \mu, r_{\mathsf{Sign}}))$
$\pi_{\mathsf{Prove}} \leftarrow\!\!\$\ \mathsf{NIZK}\{(pk, R, \mu, r_{\mathsf{Sign}}, t_{\mathsf{Prove}}); sk;$
  $f(sk) = pk,$
  $\mathsf{LIT.Vrfy}(sk, (R, \mu, r_{\mathsf{Sign}})) = 1\}$
$\pi \leftarrow (t_{\mathsf{Prove}}, \pi_{\mathsf{Prove}})$
**return** $\pi$

$\mathsf{Judge}(pk, R, \mu, \sigma, \pi_{\mathsf{Prove}})$

**if** $\mathsf{V}(\mathsf{crs}, (pk, R, \mu, r_{\mathsf{Sign}}, t_{\mathsf{Prove}}), \pi_{\mathsf{Prove}}) = 0$
  $\vee\ \mathsf{Vrfy}(R, \mu, \sigma) = 0$
  **return** $\perp$
**if** $pk \notin R$
  **return** $0$
**return** $\mathsf{Link}((R, \mu), t_{\mathsf{Sign}}, t_{\mathsf{Prove}})$

Fig. 8: Generic VRS construction.

After $\mathcal{A}$ has output a forgery $(R^*, \mu^*, \sigma^* = (r^*_{\mathsf{Sign}}, t^*_{\mathsf{Sign}}, \pi^*_{\mathsf{Sign}}))$, if $b_0 \wedge b_1 \wedge b_2$ is true, $\mathcal{B}$ uses the straight-line extractor of the NIZK to extract $(sk^*, pk^*)$ from $\pi^*_{\mathsf{Sign}}$. If $pk = pk^*$, $\mathcal{B}$ outputs $sk^*$.

By the definition of the non-invertability game and the zero-knowledgeness of the NIZK, $\mathcal{A}$ is perfectly simulated. Since $b_0$ and $b_2$ are true, we know that the straight-line extractability is successful and therefore that $f(sk^*) = pk^*$ and that $pk^* \in R^*$. Then, since $b_1$ is true, there exists a $j \in [\ell]$ such that $pk^* = pk_j$. If $j = k$, we therefore have $f(sk^*) = pk_k = pk^*$, thus $sk^*$ is a valid solution for the invertability game, and we have

$$\Pr[\mathsf{Inv}_{\mathsf{LIT},\mathcal{B}}(\lambda) = 1] = \frac{1}{\ell} \Pr[\mathsf{Forge}^{\ell}_{\Pi,\mathcal{A}}(\lambda) = 1]. \qquad \square$$

**Theorem 33.** *If the LIT has tag-indistinguishability and if the NIZK is zero-knowledge, then the VRS construction has anonymity.*

*Proof.* Let $\mathcal{A}$ be an adversary against the anonymity of the VRS. We first define an alternative security game and then construct an adversary against the tag-indistinguishability of the LIT. The alternative security game $\mathsf{Game}^b_1$

for $b \in \{0,1\}$ works like $\mathsf{Anon}^b_{\Pi,\mathcal{A}}(\lambda)$, except that during the oracle calls to $\mathsf{Sign}\mathcal{O}, \mathsf{Prove}\mathcal{O}, \mathsf{LoRSign}\mathcal{O}$, the $pi_{\mathsf{Sign}}$ and $\pi_{\mathsf{Prove}}$ are generated with the zero-knowledge simulator of the NIZK. We know that these games only differ negligibly by the zero-knowledge property of the NIZK. Then, we construct an adversary $\mathcal{B}$ against the tag-indistinguishability of the LIT. On input $(\hat{pk}_0, \hat{pk}_1)$, $\mathcal{B}$ simulates $\mathcal{A}$ in $\mathsf{Game}_1$, except for the following changes.

- Instead of generating $pk_0, pk_1$, $\mathcal{B}$ instead sets $pk_0 \leftarrow \hat{pk}_0, pk_1 \leftarrow \hat{pk}_1$.
- Whenever $\mathcal{B}$ needs to compute a tag on a message $(R, \mu, r_{\mathsf{Sign}})$ to answer an oracle query of $\mathcal{A}$, $\mathcal{B}$ instead queries its own tag oracle on input $(R, \mu, r_{\mathsf{Sign}})$.

Then, after $\mathcal{A}$ outputs a bit $\hat{b}$, $\mathcal{B}$ outputs $\hat{b}$. We know that by the definition of $\mathsf{Game}_1$ and of the tag-indistinguishability game that if $b = 0$ (or $b = 1$) then $\mathcal{A}$ is simulated as in $\mathsf{Game}^0_1$ (or $\mathsf{Game}^1_1$, respectively). Thus, we know that

$$\begin{aligned}&\left|\Pr[\mathsf{Anon}^0_{\Pi,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Anon}^1_{\Pi,\mathcal{A}}(\lambda) = 1]\right| \\ =&\left|\Pr[\mathsf{Anon}^{\mathsf{LIT}}_{\mathsf{LIT},\mathcal{B},0}(\lambda) = 1] - \Pr[\mathsf{Anon}^{\mathsf{LIT}}_{\mathsf{LIT},\mathcal{B},1}(\lambda) = 1]\right|,\end{aligned}$$

which concludes the proof. □

**Theorem 34.** *If the LIT has non-invertability and linkability and if the NIZK is straight-line extractable and zero-knowledge, then the VRS construction has accountability.*

*Proof.* Let $\ell \in \mathsf{poly}(\lambda)$ and let $\mathcal{A}$ be a *qpt* adversary against the accountability of the VRS construction. We define a series of games $\mathsf{Game}_i$, in which $\mathcal{A}$ is simulated.

$\mathsf{Game}_0$ is the same as $\mathsf{Acc}^{\ell}_{\Pi,\mathcal{A}}(\lambda)$. Define the event $\mathsf{Win}_0$ to be the event that $b_0 \wedge b_1 \wedge b_2 \wedge b_3$ is true.

$\mathsf{Game}_1$ is the same as $\mathsf{Game}_0$, except that after $\mathcal{A}$ outputs a forgery $(pk^*, R^*, \mu^*, \sigma^*, \pi^*)$ with $\sigma^* = (r^*_{\mathsf{Sign}}, t^*_{\mathsf{Sign}}, \pi^*_{\mathsf{Sign}})$ and $\pi^* = (t^*_{\mathsf{Prove}}, \pi^*_{\mathsf{Prove}})$ and $b_0 \wedge b_1 \wedge b_2 \wedge b_3$ being true, the game uses the straight-line extractor of the NIZK to extract $(sk^*_{\mathsf{Sign}}, pk^*_{\mathsf{Sign}})$ from $\pi^*_{\mathsf{Sign}}$ and $sk^*_{\mathsf{Prove}}$ from $\pi^*_{\mathsf{Prove}}$. Define the event $\mathsf{Win}_1$ to be the event that $\mathsf{Win}_0$ is true and that both extractions are successful. Then we know by the straight-line extractability of the NIZK that

$$\Pr[\mathsf{Win}_0] \leq \Pr[\mathsf{Win}_1] + \mathsf{negl}(\lambda).$$

Thus, in $\mathsf{Game}_1$ we always successfully extract if $\mathcal{A}$ wins $\mathsf{Game}_0$.

$\mathsf{Game}_2$ is the same game as $\mathsf{Game}_1$, except that we always have $pk^* \in R^*$, i.e., we define $\mathsf{Win}_2$ as $\mathsf{Win}_1 \wedge pk^* \in R^*$. We can show that the probabilities of $\mathsf{Win}_1$ and $\mathsf{Win}_2$ only differ negligibly by using the strong unforgeability of the VRS. Since $b_0$ and $b_3$ are true, $(R^*, \mu^*, \sigma^*)$ forms a valid forgery in the strong unforgeability game against the VRS. The input and oracles of $\mathcal{A}$ match in the two games. Thus, since we require non-invertability and linkability, we can construct an adversary $\mathcal{C}$ as in the proof of Theorem 32. Thus, we know that

$$\Pr[\mathsf{Win}_1] \leq \frac{1}{\ell} \Pr[\mathsf{Win}_2].$$

$\mathsf{Game}_3$ is the same as $\mathsf{Game}_2$, except that the extracted $pk^*_{\mathsf{Sign}}$ is always equal to $pk^*$, i.e., we define $\mathsf{Win}_3$ as $\mathsf{Win}_2 \wedge pk^*_{\mathsf{Sign}} = pk^*$. Again we can use the strong unforgeability of the VRS to show that $\mathsf{Game}_2$ and $\mathsf{Game}_3$ differ negligibly. With the same argument as in $\mathsf{Game}_2$, we can argue that

$$\Pr[\mathsf{Win}_2] \leq \frac{1}{\ell} \Pr[\mathsf{Win}_3].$$

We can now construct an adversary $\mathcal{B}$ against the linkability of the LIT. $\mathcal{B}$ simulates $\mathcal{A}$ as in $\mathsf{Game}_3$, i.e., we assume that $\mathcal{A}$ wins, the extraction is successful, $pk^* \in R^*$, and the extracted $pk^*_{\mathsf{Sign}} = pk^*$. Then, $\mathcal{B}$ returns $(sk^*_{\mathsf{Sign}}, sk^*_{\mathsf{Prove}}, (R^*, \mu^*, r^*_{\mathsf{Sign}}), t^*_{\mathsf{Sign}}, t^*_{\mathsf{Prove}})$ to its challenger. To argue that $\mathcal{B}$ wins the linkability game, we need to show that

1. $f(sk^*_{\mathsf{Sign}}) = f(sk^*)$
2. $\mathsf{LIT.Vrfy}(sk^*_{\mathsf{Sign}}, (R^*, \mu^*, r^*_{\mathsf{Sign}}), t^*_{\mathsf{Sign}}) = \mathsf{LIT.Vrfy}(sk^*, (R^*, \mu^*, r^*_{\mathsf{Sign}}), t^*_{\mathsf{Prove}}) = 1$
3. $\mathsf{Link}((R^*, \mu^*, r^*_{\mathsf{Sign}}), t^*_{\mathsf{Sign}}, t^*_{\mathsf{Prove}}) = 0.$

Condition (2) follows immediately from the successful extraction, while this fact together with the assumption that $pk^*_{\mathsf{Sign}} = pk^*$ implies (1). For condition (3), we know that since $b_1$ is true, the judge outputs 0. Due to $pk^* \in R^*$, we know that this only happens if condition (3) is true. Thus, we know that if $\mathcal{A}$ wins $\mathsf{Game}_3$, then $\mathcal{B}$ wins the the linkability game. Therefore, we have that

$$\Pr[\mathsf{Acc}^{\ell}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \frac{1}{\ell^2} \Pr[\mathsf{Linkable}^{\mathsf{LIT}}_{\mathsf{LIT},\mathcal{B}}(\lambda) = 1] + \mathsf{negl}(\lambda),$$

and the VRS is accountable[5].

**Theorem 35.** *If the LIT is unforgeable, and if the NIZK is straight-line extractable and zero-knowledge, then the VRS construction has non-seizability.*

*Proof.* Let $\mathcal{A}$ be a *qpt* adversary against the accountability of the VRS construction. We construct an adversary $\mathcal{B}$ against the unforgeability of the LIT. On input $pk$, $\mathcal{B}$ simulates $\mathcal{A}$ as in the non-seizability game, except that $\mathcal{B}$ sets $pk_0 \leftarrow pk$, and except for the following two changes.

- When $\mathcal{A}$ makes a query $(i, R, \mu)$ to its $\mathsf{Sign}$ oracle with $i = k$, $\mathcal{B}$ generates the tag with $t_{\mathsf{Sign}} \leftarrow\!\!\$\ \mathsf{Tag}\mathcal{O}((R, \mu, r_{\mathsf{Sign}}))$, where $r_{\mathsf{Sign}} \leftarrow\!\!\$\ \{0,1\}^{\lambda}$ as in the construction. The proof $\pi$ is generated with the simulator of the NIZK.
- When $\mathcal{A}$ makes a query $(i, R, \mu, \sigma)$ to its $\mathsf{Prove}\mathcal{O}$ oracle with $i = k$, and if $pk \in R$ and $\mathsf{Vrfy}(R, \mu, \sigma) = 1$, and where $\sigma = (r_{\mathsf{Sign}}, t_{\mathsf{Sign}}, \pi_{\mathsf{Sign}})$, then $\mathcal{B}$ generates the tag with $t_{\mathsf{Prove}} \leftarrow\!\!\$\ \mathsf{Tag}\mathcal{O}((R, \mu, r_{\mathsf{Sign}}))$. The proof is generated with the simulator of the NIZK.

[5] It is possible to reduce the multiplicative loss from $1/\ell^2$ to $1/\ell$ by going from $\mathsf{Game}_1$ to $\mathsf{Game}_3$ directly.

When $\mathcal{A}$ outputs a forgery $(R^*, mu^*, \sigma^* = (r^*_{\mathsf{Sign}}, t^*_{\mathsf{Sign}}, \pi^*_{\mathsf{Sign}}))$ and if $b_0 \wedge b_1 \wedge b_2 \wedge b_3$ is true, $\mathcal{B}$ uses the straight-line extractor of the NIZK to extract $(sk^*, pk^*)$ from $\pi^*_{\mathsf{Sign}}$. Then, $\mathcal{B}$ queries $t^*_{\mathsf{Prove}} \leftarrow\$ \mathsf{Tag}\mathcal{O}((R^*, \mu^*, r^*_{\mathsf{Sign}}))$ and uses the simulator to create $\pi^*_{\mathsf{Prove}}$ to define $\pi^* = (t^*_{\mathsf{Prove}}, \pi^*_{\mathsf{Prove}})$. Afterwards, $\mathcal{B}$ outputs $(sk^*, (R^*, \mu^*, r^*_{\mathsf{Sign}}), t^*_{\mathsf{Sign}})$.

By definition of the non-invertability game of the LIT and the zero-knowledge of the NIZK, we know that $\mathcal{A}$ is perfectly simulated. Since $b_1$ is true, $pk_0 \in R^*$. Furthermore, we know that $\mathsf{Judge}(pk_0, R^*, \mu^*, \sigma^*, \pi^*_{\mathsf{Prove}}) = \perp$ if either $\mathsf{NIZK.Vrfy}(\mathsf{crs}, (pk_0, R^*, t^*_{\mathsf{Prove}}), \mu^*, \pi^*_{\mathsf{Prove}}) = 0$ or $\mathsf{Vrfy}(R^*, \mu^*, \sigma^*) = 0$. However, the former case does not happen due to the correctness of the NIZK, while the latter is not true due to $b_0$ being true. Thus, $\mathsf{Judge}(pk_0, R^*, \mu^*, \sigma^*, \pi^*_{\mathsf{Prove}}) = 1$, and $\mathsf{Link}((R^*, \mu^*), t^*_{\mathsf{Sign}}, t^*_{\mathsf{Prove}}) = 1$. Due to this and $b_0$ being true, we know that $\mathcal{B}$ wins the unforgeability game and we have

$$\Pr[\mathsf{Forge}^{\mathsf{LIT}}_{\mathsf{LIT},\mathcal{B}}(\lambda) = 1] = \Pr[\mathsf{Seiz}_{\Pi,\mathcal{A}}(\lambda) = 1]. \qquad \square$$

We now want to instantiate the generic construction with lattice-based building blocks. For the LIT, we can use the one described in Lemma 18. The NIZK need not only be lattice-based, but also be able to prove the statements we need. We use the NIZK from [35] made straight-line extractable with Katsumata's transform [26] as seen in [9, 10]. We need to able to prove that

1. $f(sk) = pk$
2. $pk \in R$
3. $\mathsf{LIT}, \mathsf{Vrfy}(sk, (R, \mu, r_{\mathsf{Sign}}), t_{\mathsf{Sign}}) = 1$.

Since $R, \mu, r_{\mathsf{Sign}}$ are public, we can use the argument from [9] to show that we can prove (1) and (3) with the NIZK. For (2), in our use case of SSS the ring $R$ will only consist of two public keys. Thus, it is sufficient if we prove (2) by creating an OR-proof over the condition in (1). If one wants to use the VRS with more than two parties, it is advisable to prove (2) by using accumulators (e.g. [32]) or one-out-of-many proofs (e.g. [34]) to be more efficient.

## 4 Sanitizable Signature Schemes

We now move on to defining sanitizable signature schemes in this section before providing instantiations in the next section.

The definition for sanitizable signature schemes is an extension of regular signature schemes. Due to the addition of the sanitizer, extra operations for generating keys for the sanitizer as well as sanitization itself are required.

**Definition 36.** *A* sanitizable signature scheme $\mathsf{SSS}$ *is a tuple of seven probabilistic polynomial-time algorithms* $\mathsf{SSS} = (\mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Vrfy}, \mathsf{Proof}, \mathsf{Judge})$ *defined as follows:*

$(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}}) \leftarrow\$ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$: *The algorithm takes the security parameter as input and outputs a signer key pair* $(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}})$.

$(pk_{\mathsf{San}}, sk_{\mathsf{San}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{San}}(1^\lambda)$: *The algorithm takes the security parameter as input and outputs a sanitizer key pair* $(pk_{\mathsf{San}}, sk_{\mathsf{San}})$.

$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$: *On input of a message* $\mu$ *in the message space* $\mathcal{M}$*, signer secret key* $sk_{\mathsf{Sig}}$*, a sanitizer public key* $sk_{\mathsf{San}}$*, as well as the admissible sanitization rights* ADM*, the algorithm outputs a signature* $\sigma$ *or an error message* $\perp$*.* ADM *contains the indices of block that are admissible for modification. Furthermore, we assume that* ADM *is always valid with regards to the input message.* $\mathrm{ADM}_0 \cap \mathrm{ADM}_1$ *denotes the intersection of admissible blocks.*

$(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{Sanit}(\mu, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk_{\mathsf{Sig}})$: *On input of a message* $\mu$*, a signature* $\sigma$*, a sanitizer secret key* $sk_{\mathsf{San}}$*, a signer public key* $pk_{\mathsf{Sig}}$*, as well as the modification instructions* MOD*. The algorithm outputs the modified message* $m'$ *along with the corresponding sanitized signature* $\sigma'$ *or an error message* $\perp$*. We model* MOD *as a function which takes the old message as input and outputs the modified message* $m' \leftarrow \mathrm{MOD}(m)$*. We write* $\mathrm{MOD}(\mathrm{ADM}) \rightarrow \top/\perp$ *to check if the intended modifications are allowed or not.*

$d \leftarrow \mathsf{Vrfy}(\mu, \sigma, pk_{\mathsf{Sig}}, pk_{\mathsf{San}})$: *On input of a message* $\mu$*, a signature* $\sigma$*, a signer public key* $pk_{\mathsf{Sig}}$*, and a sanitizer public key* $pk_{\mathsf{San}}$*, the algorithm outputs a decision bit* $d \in \{\top, \perp\}$*.*

$\pi \leftarrow\!\!\$\ \mathsf{Proof}(\mu, \sigma, \{(\mu_i, \sigma_i)\}_{i=1}^k, sk_{\mathsf{Sig}}, pk_{\mathsf{Sig}}, pk_{\mathsf{San}})$: *On input of a message* $\mu$*, a signature* $\sigma$*, a set of additional message-signature pairs* $\{(\mu_i, \sigma_i)\}_{i=1}^k$*, signer secret key* $sk_{\mathsf{Sig}}$ *and public key* $pk_{\mathsf{Sig}}$*, and the sanitizer public key* $pk_{\mathsf{San}}$*, the algorithm outputs a proof* $\pi \in \{0,1\}^*$ *or an error message* $\perp$*.*

$d \leftarrow \mathsf{Judge}(\mu, \sigma, pk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \pi)$: *On input of a message* $\mu$*, a signature* $\sigma$*, a signer public key* $pk_{\mathsf{Sig}}$*, a sanitizer public key* $pk_{\mathsf{San}}$*, and a proof* $\pi$*, the algorithm outputs a bit determining who generated the signature, i.e.,* $d \in \{\mathsf{Sig}, \mathsf{San}\}$*, or returns an error message* $\perp$*.*

We follow the correctness definition of Brzuska et al. [11, 12] as well as subsequent works that require that genuinely signed or sanitized messages are accepted, and a genuinely generated proof by the signer will lead the judge to determine the accountable party correctly. The formal definition can be found in Appendix B, Definition 72.

The definition for blocks admissible for modification, ADM, and description of desired modifications, MOD, also follows from [11]. MOD is a list of $t$ binary values $\{0,1\}^t$, where $t$ is the block length of the message. The value at the $i$-th position of ADM determines whether block $i$ is admissible for modification. $\mathrm{ADM} \subseteq \mathbb{N} \times \mathcal{M}$ describes the desired modification. The first part of the tuple indicates the block to be modified while the second part is the new content.

### 4.1 Security Notions

Here we briefly describe the intuition for each of the usual security properties for sanitizable signature schemes. Observe that most of these properties were stated in [11, 12, 14, 16]. Furthermore, some later papers like [29] consider "stronger"

versions of the security properties while others like [12, 30, 23] introduce slightly weaker versions. In this work, we will adhere to the basic notions for security.

In the following list we will state all possible security notions for sanitizable signature schemes and give a brief and informal intuition on what they achieve. Formal definitions for all notions can be found in Appendix B.

**Unforgeability** No efficient adversary should be able to produce valid message-signature pairs that it has not seen before.

**Immutability** A malicious sanitizer should not be able to modify blocks that the signer has not set to be admissible for modification.

**Signer Accountability** A signer must not be able to craft a valid message-signature pair and a proof, such that the $\mathsf{Judge}$ algorithm will output $\mathsf{San}$.

**Sanitizer Accountability** A sanitizer must not be able to craft a valid message-signature pair such that honest proofs generated by the signer will lead to the $\mathsf{Judge}$ algorithm outputting $\mathsf{Sig}$.

**Accountability** Both sanitizer- and signer accountability hold.

**Non-Interactive Public Accountability** Given a valid pair of message and signature, a third party can correctly determine who created the signature with no additional information.

**Transparency** Detecting if a signature has been sanitized must be hard.

**Privacy** Given a sanitized message, it must be hard to recover any information about the original message's content before being sanitized.

**Unlinkability** Two different sanitized messages cannot be identified as belonging to the same initial message.

**Invisibility** It must be hard to detect which parts of the message may be modified by the sanitizer.

In the literature, there exist several varieties of each of these properties, the use of which we try to limit in this paper for improved readability. To this end, we refer to the notion formally known as "proof-restricted transparency" simply as "transparency". Definitions for strong/weak variants can be found in Appendix B.

### 4.2 Signer Accountability

For the main body we limit ourselves to the formal definition of signer accountability, as we later show a new attack on this notion in Theorem 46 for one of the constructions examined as part of this work.

**Definition 37 (Signer-Accountability).** *A sanitizable signature scheme* $\mathsf{SSS}$ *is (strongly) signer-accountable, if for all qpt adversaries* $\mathcal{A}$*, the advantage in winning the game* $\mathsf{Exp}^{\mathsf{(s)sig\text{-}acc}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$ *as described in Fig. 9, defined as*

$$\mathsf{Adv}^{\mathsf{(s)sig\text{-}acc}}_{\mathsf{SSS},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Exp}^{\mathsf{(s)sig\text{-}acc}}_{\mathsf{SSS},\mathcal{A}}(\lambda) = 1\right]$$

*is negligible in the security parameter* $\lambda$*.*

$\mathsf{Exp}^{\text{(s)sig-acc}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$

$(pk_{\mathsf{San}}, sk_{\mathsf{San}}) \leftarrow\!\$\ \mathsf{KGen}_{\mathsf{San}}(1^\lambda)$
$\mathcal{Q}_{\mathsf{San}} \leftarrow \emptyset$
$(pk^*_{\mathsf{Sig}}, \pi^*, \mu^*, \sigma^*) \leftarrow\!\$\ \mathcal{A}^{\mathsf{Sanit}\mathcal{O}}(pk_{\mathsf{San}})$
**if** $(*, \mu^*, \sigma^*, pk^*_{\mathsf{Sig}}) \notin \mathcal{Q}_{\mathsf{San}}$
$\quad \wedge\ \mathsf{Vrfy}(\mu^*, \sigma^*, pk^*_{\mathsf{Sig}}, pk_{\mathsf{San}}) = \top$
$\quad \wedge\ \mathsf{Judge}(\mu^*, \sigma^*, pk^*_{\mathsf{Sig}}, pk_{\mathsf{San}}, \pi^*) = \mathsf{San}$
$\quad$ **return** 1
**else return** 0

$\mathsf{Sanit}\mathcal{O}(sk_{\mathsf{San}}, \cdot, \cdot, \cdot, \cdot)$
$\quad$ with $(\mu, \mathrm{MOD}, \sigma, pk_{\mathsf{Sig}})$

$(\mu', \sigma') \leftarrow\!\$\ \mathsf{Sanit}(\mu, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk_{\mathsf{Sig}})$
$\mathcal{Q}_{\mathsf{San}} \leftarrow \mathcal{Q}_{\mathsf{San}} \cup \{(\mu, \mu', \sigma', pk_{\mathsf{Sig}})\}$
**return** $(\mu', \sigma')$

Fig. 9: Game based security definition for signer accountability. Inclusion of the gray box yields strong signer accountability.

## 5 Lattice-Based Instantiations with CH

This section is devoted to the instantiation of the various SSS constructions with the chameleon hash function. For other types of constructions, see Appendices D and E. We explain the results of utilizing the chameleon hash function we constructed in Section 3.1 first. Due to the missing uniqueness of the chameleon hash function, the resulting sanitizable signature schemes lack important security features like unforgeability and accountability.[6] Luckily, we can employ the transform by [15]. This transform takes a sanitizable signature scheme, which does not necessarily satisfy unforgeability, and a verifiable ring signature (VRS) to a sanitizable signature scheme that satisfies unforgeability and full accountability. In the present case, we make use of the VRS constructed in Section 3.2, to increase the security guarantees of the lattice-based SSSs.

Before we iterate through the collection of SSS constructions using chameleon hash functions, we explain the transform to ensure unforgeability and accountability from [15].

**Theorem 38.** *For a sanitizable signature scheme* $\mathsf{SSS}_2$ *resulting from applying the transformation explained in [15, Fig. 5] to a sanitizable signature scheme* $\mathsf{SSS}_1$ *and a verifiable ring signature* VRS*, the following implications hold: If* $\mathsf{SSS}_1$ *is weakly immutable, then* $\mathsf{SSS}_2$ *is immutable. If* $\mathsf{SSS}_1$ *is weakly unlinkable, and* VRS *is strongly unforgeable, then* $\mathsf{SSS}_2$ *is unlinkable. If* $\mathsf{SSS}_1$ *is strongly invisible then* $\mathsf{SSS}_2$ *is strongly invisible. If* VRS *is strongly accountable, then* $\mathsf{SSS}_2$ *is strongly signer accountable. If* VRS *is strongly non-seizable, then* $\mathsf{SSS}_2$ *is strongly sanitizer accountable. If* VRS *is anonymous and* $\mathsf{SSS}_1$ *is strongly transparent, then* $\mathsf{SSS}_2$ *is strongly transparent.*

[6] Only in [3], sanitizer accountability has not been defined and has neither been attacked nor proven.

Finally, we note that standard instantiations of PRFs and PRGs are used in the protocols, without further specification. Post-quantum secure variants are given, for instance, in [25].

### 5.1 The ACMT05 Construction

The work by Ateniese et al. [3] marked the beginning of sanitizable signatures and gave the first construction of such. This first construction makes use of only two building blocks, a secure digital signature scheme $\Sigma$ and a chameleon hash function $\mathsf{CH}$ which is strongly unforgeable. While the strongly unforgeable notion is not defined explicitly, we inferred that full collision-resistance and indistinguishability is sufficient.

The sanitizable signature scheme in [3] is stated to be *indistinguishable*, *unforgeable*, and satisfy *identical distribution of sanitized and original signatures*, [3, Section 4.4]. By the general relations between the notions, we have the following results.

**Corollary 39.** *The construction by Ateniese et al. in [3] instantiated using a DSS from* $\{\mathsf{Dilithium}, \mathsf{Falcon}\}$ *and the chameleon hash function constructed in Fig. 7 satisfies unforgeability, signer accountability, immutability, transparency, and privacy.*

We note that sanitizer accountability and invisibility have been developed after the publication of the work by Ateniese et al. The corresponding construction has not been analyzed regarding this notion. Here, we take the easy road and apply Theorem 38, to additionally achieve sanitizer accountability.

**Corollary 40.** *Using the transform from Theorem 38 and combining the verifiable ring signature given in Fig. 8 with the construction by Ateniese et al. in [3] as instantiated in Corollary 39, we obtain a sanitizable signature that satisfies unforgeability, signer accountability, sanitizer accountability, immutability, transparency, and privacy.*

### 5.2 The LZCS16−1 Construction

Recall that [30] gives two constructions. One is based on accountable ring signatures, (cf. Appendix E) while the other uses rerandomizable tagging schemes which are constructed from tag-based trapdoor functions and double-trapdoor chameleon hash functions. While tag-based trapdoor functions are merely weaker versions of chameleon hash functions, and thus, can be instantiated from our lattice-based chameleon hash function given in Fig. 7, it is not clear whether double-trapdoor chameleon hash functions can be instantiated from lattices. Thus, as of now, the construction by Lai et al. using chameleon hash functions, cannot be instantiated from known lattice constructions.

### 5.3 The BCD$^+$17 Construction

The construction by Beck et al. [6] uses a regular digital signature scheme, a *labeled* PKE, and a secure chameleon hash function, by which they mean indistinguishable, unique, and standard collision-resistant. Further, the sanitizable signature construction requires a $\mathsf{PRG}$ and a $\mathsf{PRF}$.

As labeled PKEs are non-standard, we present a brief account including how to construct them from regular PKEs. In the construction, the labeled PKE provides strong invisibility of the sanitizable signature scheme, in which the signer public key is used as a label to prevent re-use of ciphertexts from different signer public keys. Further details including full definitions and proofs of the following can be found in Appendix A.5.

**Labeled PKEs from regular PKEs.** A *labeled PKE* (we refer to Appendix A.5) consists of a triple $(\mathsf{KGen}_\tau, \mathsf{Enc}_\tau, \mathsf{Dec}_\tau)$, closely resembling the properties of regular PKEs. The key generation algorithm $\mathsf{KGen}_\tau$ creates a key pair $(pk, sk)$. The probabilistic encryption algorithm $\mathsf{Enc}_\tau$ additionally requires a label $\tau \in \{0,1\}^n$, where the length $n$ of the label depends on the security parameter $\lambda$. The decryption algorithm $\mathsf{Dec}_\tau$ additionally requires the label $\tau$. The (perfect) correctness of a labeled PKE is defined as $\mathsf{Enc}_\tau(sk, \mathsf{Dec}_\tau(pk, \mu, \tau), \tau) = \mu$ whenever $(pk, sk) \leftarrow\!\!\$\ \mathsf{KGen}_\tau$ is honestly generated.

We explain now, how IND-CCA secure PKEs can be transformed to IND-CCA secure labeled PKEs with the proof provided in Appendix A.5. Given a PKE $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, we define a labeled PKE $\Pi_\tau = (\mathsf{KGen}_\tau, \mathsf{Enc}_\tau, \mathsf{Dec}_\tau)$ as follows. First, we note that for any label set $\{0,1\}^n$ we can concatenate a label $\tau$ and a message $\mu$ as $\tau\|\mu$ so that both, the label and the message can be recovered from $\tau\|\mu$ by either projecting to the first $n$ bits or removing them. Then, $\mathsf{KGen}_\tau = \mathsf{KGen}$ and $\mathsf{Enc}_\tau(pk, \mu, \tau) \leftarrow \mathsf{Enc}(pk, \tau\|\mu)$. Finally, the decryption $\mathsf{Dec}_\tau$ on input $(sk, c, \tau)$ first runs $\mathsf{Dec}(sk, c)$ to get $\tau'\|\mu'$. Then, $\mathsf{Dec}_\tau$ checks if $\tau'$ coincides with the input label $\tau$. If so, $\mathsf{Dec}_\tau$ returns $\mu'$. Otherwise, it returns $\perp$.

**Instantiation of the SSS construction.** The uniqueness, which our construction in Section 3.1 does not achieve, is required for unforgeability, signer accountability, and sanitizer accountability. So that we have the following result that follows from [6, Theorem 1] and the construction given in Section 3.1.

**Corollary 41.** *The SSS construction by Beck et al. [6] instantiated with a signature scheme among* $\{\mathsf{Dilithium}, \mathsf{Falcon}\}$*, the PKE* $\mathsf{Kyber}$*, and the chameleon hash function given in Fig. 7, satisfies immutability, transparency, privacy, and strong invisibility.*

The missing unforgeability and signer and sanitizer accountability are added via the transform Theorem 38.

**Corollary 42.** *The transform in Theorem 38 applied with the VRS given in Fig. 8 to the construction by Beck in [6] as instantiated in Corollary 41, satisfies unforgeability, signer accountability, sanitizer accountability, immutability, transparency, privacy, and strong invisibility.*

### 5.4 The CDK$^+$17 Construction

In [16], Camenisch et al. introduce a new notion called chameleon hash function with ephemeral trapdoors (CHET). They show how to generically construct CHET from chameleon hash functions such that the required security properties carry over. Thus, it is sufficient to have a chameleon hash function with the required properties. In total, the requirements are a correct, indistinguishable, unique, and standard collision-resistant chameleon hash function, a secure DSS, an IND-CPA secure PKE, PRGs, and PRFs, and an indistinguishable and standard collision-resistant CHET.

In their construction, Camenisch et al. use the uniqueness of the CHF in the proof of unforgeability and sanitizer accountability. Hence, from [16, Theorem 3] and taking into account the missing uniqueness, we get the following.

**Corollary 43.** *The construction by Camenish et al. in [16] instantiated using a DSS from* {Dilithium, Falcon}*, the PKE* Kyber*, the chameleon hash function given in Fig. 7, which is used twice, for the chameleon hash function and the CHET, satisfies signer accountability, immutability, transparency, privacy, and invisibility.*

To ensure unforgeability and sanitizer accountability, we apply Theorem 38.

**Corollary 44.** *The transform in Theorem 38 applied with the VRS given in Fig. 8 to the construction by Camenisch et al. in [16] as instantiated in Corollary 43, satisfies unforgeability, signer accountability, sanitizer accountability, immutability, transparency, privacy, and invisibility.*

### 5.5 The BFF$^+$09 Construction

Brzuska et al. [11] give a SSS construction that relies on a chameleon hash function but—unlike the constructions in [6] and [16]—they do not require the chameleon hash function to satisfy the uniqueness property. It therefore is an interesting candidate to be instantiated using the new construction in Section 3.1, without the need for the transform in Theorem 38. However, it turns out that an error in the security proof makes the signer accountability of the construction vulnerable to a generic attack. Therefore, this section is structured differently than the prior ones. First, we show that the security formalization of tagged chameleon hashes is unachievable and needs to be replaced in the construction by the tagged version of full collision resistance. Afterwards, we present the attack on signer accountability, which works irrespectively of the underlying building blocks. Still, the construction by Brzuska et al. can be instantiated with the chameleon hash function we constructed, merely missing the signer accountability. To achieve signer accountability as well, we can apply the transform in Theorem 38.

**Idea of the Construction.** We begin with a short description of the construction given in [11]. The full construction can be found in Appendix C.2.

First, the message is divided into blocks. Admissible blocks are hashed using a tagged chameleon hash function. The signer computes the tag as the output of a pseudorandom function with a random input value and provides this random input as proof for the accountability. It then signs the output of the tagged chameleon hash function. The sanitizer can use its private key to find collisions on the chameleon hash function, however it chooses the tag randomly. While this ensures sanitizer accountability, we show that signer-accountability for this construction is unachievable, contrary to the claim in [11]. Indeed, there are two distinct errors in the presentation of [11]. First, the notion of *collision-resistance under random-tagging attacks*, which is a strong version of collision-resistance for chameleon hash functions, is too strong and in fact, cannot be achieved by any chameleon hash function. The reason is the oracle being too strong and the winning condition too weak. We present the generic attack here in detail. Second, the claimed signer-accountability does not hold independently of the chameleon hash function's security properties. We present an outline of the generic attack at the end of this section and the detailed proof in Appendix F.3.

**Infeasibility of Random Tagging.** To show security, Brzuska et al. [11] rely on different properties for the different SSS security notions. For signer accountability, they rely on the *collision-resistance under random-tagging attacks* of the used chameleon hash function. This security notion was developed along with the SSS construction and we describe it in Fig. 10.

$\mathsf{RndTag}^{\mathsf{CH}}_{\mathcal{A}}(\lambda)$

$(pk, sk) \leftarrow\!\!\$\ \mathsf{CKGen}(1^\lambda)$
$(\mathrm{TAG}, \mu, r, \mathrm{TAG}', \mu', r') \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Adapt}\mathcal{O}}(pk)$
**return** 1 **if** $(\mathrm{TAG}, \mu) \neq (\mathrm{TAG}', \mu')$
$\wedge\ \mathsf{CH}(pk, \mathrm{TAG}, \mu, r) = \mathsf{CH}(pk, \mathrm{TAG}', \mu', r')$
$\wedge\ \{(\mathrm{TAG}, \mu), (\mathrm{TAG}', \mu')\} \neq \{(\mathrm{TAG}_i, \mu_i), (\mathrm{TAG}'_i, \mu'_i)\},$ for all $i = 1, \ldots, q$
$\wedge\ \{(\mathrm{TAG}, \mu), (\mathrm{TAG}', \mu')\} \neq \{(\mathrm{TAG}'_i, \mu'_i), (\mathrm{TAG}'_j, \mu'_j)\},$ for all $i, j = 1, \ldots, q$

$\mathsf{Adapt}\mathcal{O}(sk, \mathrm{TAG}_i, \mu_i, r_i, \mu'_i)$

$\mathrm{TAG}'_i \leftarrow\!\!\$\ \{0,1\}^{2n}$
$r'_i \leftarrow\!\!\$\ \mathsf{CAdapt}(sk, \mathrm{TAG}_i, \mu_i, r_i, \mathrm{TAG}'_i, \mu'_i)$
Store $(\mathrm{TAG}_i, \mu_i), (\mathrm{TAG}'_i, \mu'_i)$
**return** $(\mathrm{TAG}'_i, r'_i)$

Fig. 10: Collision-resistance under random-tagging attacks from [11].

To exclude trivial wins, the game numbers the adversary's queries and stores them as $(\mathrm{TAG}_i, \mu_i)$ and $(\mathrm{TAG}'_i, \mu'_i)$, for $i = 1, \ldots, q$. An adversary wins, if it can

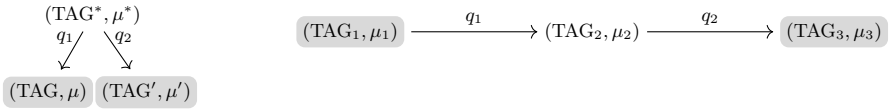


Fig. 11: Left: a trivial collision that is excluded in the security game. Right: a trivial collision that composes a generic attack on the security notion.

output a non-trivial collision $\mathsf{CH}(pk, \mathrm{TAG}, \mu, r) = \mathsf{CH}(pk, \mathrm{TAG}', \mu', r')$ with

$$\begin{aligned} &\{(\mathrm{TAG}, \mu), (\mathrm{TAG}', \mu')\} \neq \{(\mathrm{TAG}_i, \mu_i), (\mathrm{TAG}'_i, \mu'_i)\} \text{ for all } i, \\ \text{and} \quad &\{(\mathrm{TAG}, \mu), (\mathrm{TAG}', \mu')\} \neq \{(\mathrm{TAG}'_i, \mu'_i), (\mathrm{TAG}'_j, \mu'_j)\} \text{ for all } i, j. \end{aligned}$$

The former condition prevents the adversary to output a collision that it obtains from a single query to its oracle $\mathsf{Adapt}\mathcal{O}$. The latter prevents the adversary trivial transitive collisions: By querying the oracle $\mathsf{Adapt}\mathcal{O}$ twice on the same input for different target messages, as depicted on the left of Fig. 11.

It turns out, that the two excluded types of trivial collisions do not suffice. In fact, the adversary can make an arbitrary query to the oracle $\mathsf{Adapt}\mathcal{O}$ and then make a second query on the response of the first query. Then, all three tuples collide, but the first and third as output, are not defined as a trivial win. This is depicted on the right of Fig. 11. In conclusion, *no* chameleon hash function can satisfy this security notion as stated in the theorem below.

**Theorem 45.** *Collision-resistance under random-tagging attacks is unachievable for any chameleon hash function.*

*Proof.* The attack idea is depicted in Fig. 11. Given a public key $pk$ we define the attacker against collision-resistance under random-tagging attacks as follows: First, the adversary picks distinct messages $\mu_1$, $\mu_2$, and $\mu_3$. Further it generates $\mathrm{TAG}_1$ and $r_1$. Then, the adversary makes two queries to the oracle: first, query $(\mathrm{TAG}_1, \mu_1, r_1, \mu_2)$ to receive $(\mathrm{TAG}_2, r_2)$, second, $(\mathrm{TAG}_2, \mu_2, r_2, \mu_3)$ to receive $(\mathrm{TAG}_3, r_3)$. Thus, we have a collision of the hash values of $(pk, \mathrm{TAG}_1, \mu_1, r_1)$ and $(pk, \mathrm{TAG}_3, \mu_3, r_3)$, the definition of the oracle. The adversary returning $(\mathrm{TAG}_1, \mu_1, r_1)$ and $(\mathrm{TAG}_3, \mu_3, r_3)$ wins the game. Indeed, the collision is valid, as $(\mathrm{TAG}_1, \mu_1)$ was never an output (hence the collision is not of the second form), and $(\mathrm{TAG}_1, \mu_1)$ and $(\mathrm{TAG}_3, \mu_3)$ were not part of the same query. □

**Signer-Accountability.** Theorem 45 leaves a gap in the construction of [11] as no instantiation achieves signer accountability. A natural question that arises is whether the construction is secure if one strengthens the requirements towards the underlying chameleon hash function by excluding the transitivity attack that we described above. Here, we answer this question in the negative by providing a generic attack against the signer accountability.

We show that a signer can put the blame on a message to the sanitizer even though the sanitizer did not sanitize to this message. On a high-level, the attack

works as follows. The signer prepares an arbitrary message containing two blocks $\mu_1 \| \mu_2$ such that only the second one is admissible. It then lets the sanitizer change the message to $\mu_1 \| \mu_2^*$. Finally, the adversary can create a signature on the message $\mu_1^* \| \mu_2^*$ by replacing the first message block. When questioned about the message $\mu_1^* \| \mu_2$ created by the signer, the judge will blame the sanitizer. This is stated formally in the following theorem, the proof is given in Appendix F.3.

**Theorem 46.** *There is an efficient adversary $\mathcal{A}$ that breaks signer accountability of the* BFF$^+$09 *construction with probability* 1*, using a single query to* Sanit$\mathcal{O}$*.*

**Instantiations.** Finally, we want to apply our constructions in Section 3 to the construction by Brzuska et al. To account for the unachievable security notion for tagged chameleon hash functions, we refer to Appendix C.1 for the notion of f-CR tagged chameleon hash functions, which can be constructed from a f-CRCHash as in Fig. 7. As uniqueness of the CHF is not required, we get the following result.

**Corollary 47.** *The construction by Brzuska et al. in [11] instantiated using a DSS from* {Dilithium, Falcon} *and the tagged chameleon hash function deduced from the chameleon hash in Fig. 7 as explained in Appendix C.1 satisfies unforgeability, sanitizer accountability, immutability, transparency, and privacy.*

This follows from [11, Theorem 5.3], again noting that signer accountability does not hold. Using Theorem 38, we can increase the security to cover signer accountability.

**Corollary 48.** *The transform in Theorem 38 applied with the VRS given in Fig. 8 to the construction by Brzuska et al. in [11] as instantiated in Corollary 47, satisfies unforgeability, signer accountability, sanitizer accountability, immutability, transparency, and privacy.*

### 5.6 General Observations

Unfortunately, the efficiency of the proposed instantiations may be suboptimal for any real world use cases. For example, the signature of the construction of [16] instantiated securely with our building blocks and the transform of [15] contains an encryption of a chameleon hash trapdoor, i.e. an encryption of a G-trapdoor $\mathbf{R}$ for each message block. On the positive side, we do not impose inefficient requirements such as a superpolynomial modulus and the reduction losses are not substantial.

**Corollary 49.** *There exists an SSS construction that has unforgeability, signer accountability, sanitizer accountability, immutability, transparency, privacy, and invisibility in the random oracle model, if LWE and LWR and SIS with polynomial modulus are hard.*

Furthermore, we expect our building blocks and thus the generic constructions of SSS to carry over to ring or module lattices in a straightforward fashion.

An additional drawback is the necessary inclusion of the transform of [15] for many SSS constructions, which incurs the cost of a NIZK. However, a secure construction without this transformation requires uniqueness of the chameleon hash function, which seems highly non-trivial for lattice-based constructions. Previous non lattice-based constructions used bijections to ensure that only one auxiliary value exists, thus even with a trapdoor an adversary cannot find a second auxiliary value to break uniqueness. Lattice-based trapdoored functions, especially ones used in conjunction with PreSample, have many different preimages for each image by design, which leads us to believe that a chameleon hash achieving uniqueness requires a fundamentally different approach to trapdoor its hash function.

## References


[1] Miklós Ajtai. "Generating Hard Instances of Lattice Problems (Extended Abstract)". In: *28th ACM STOC*. ACM Press, May 1996, pp. 99–108. DOI: 10.1145/237814.237838.

[2] Nabil Alkeilani Alkadri, Poulami Das, Andreas Erwig, Sebastian Faust, Juliane Krämer, Siavash Riahi, and Patrick Struck. "Deterministic Wallets in a Quantum World". In: *ACM CCS 2020*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. ACM Press, Nov. 2020, pp. 1017–1031. DOI: 10.1145/3372297.3423361.

[3] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. "Sanitizable Signatures". In: *ESORICS 2005*. Ed. by Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann. Vol. 3679. LNCS. Springer, Berlin, Heidelberg, Sept. 2005, pp. 159–177. DOI: 10.1007/11555827_10.

[4] Abhishek Banerjee, Chris Peikert, and Alon Rosen. "Pseudorandom Functions and Lattices". In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Berlin, Heidelberg, Apr. 2012, pp. 719–737. DOI: 10.1007/978-3-642-29011-4_42.

[5] Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. "More Efficient Commitments from Structured Lattice Assumptions". In: *SCN 18*. Ed. by Dario Catalano and Roberto De Prisco. Vol. 11035. LNCS. Springer, Cham, Sept. 2018, pp. 368–385. DOI: 10.1007/978-3-319-98113-0_20.

[6] 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: *ACISP 17, Part I*. Ed. by Josef Pieprzyk and Suriadi Suriadi. Vol. 10342. LNCS. Springer, Cham, July 2017, pp. 437–452. DOI: 10.1007/978-3-319-60055-0_23.

[7] David Bernhard, Georg Fuchsbauer, Essam Ghadafi, Nigel P. Smart, and Bogdan Warinschi. "Anonymous attestation with user-controlled linkability". In: *Int. J. Inf. Sec.* 12.3 (2013), pp. 219–249. DOI: 10.1007/S10207-013-0191-Z. URL: https://doi.org/10.1007/s10207-013-0191-z.

[8] Ward Beullens, Samuel Dobson, Shuichi Katsumata, Yi-Fu Lai, and Federico Pintore. "Group Signatures and More from Isogenies and Lattices: Generic, Simple, and Efficient". In: *EUROCRYPT 2022, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. LNCS. Springer, Cham, 2022, pp. 95–126. DOI: 10.1007/978-3-031-07085-3_4.

[9] Johannes Blömer, Jan Bobolz, and Laurens Porzenheim. "A Generic Construction of an Anonymous Reputation System and Instantiations from Lattices". In: *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part II*. Ed. by Jian Guo and Ron Steinfeld. Vol. 14439. Lecture Notes in Computer Science. Springer, 2023, pp. 418–452. DOI: 10.1007/978-981-99-8724-5_13. URL: https://doi.org/10.1007/978-981-99-8724-5_13.

[10] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Alessandro Sorniotti. "A Framework for Practical Anonymous Credentials from Lattices". In: *CRYPTO 2023, Part II*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14082. LNCS. Springer, Cham, Aug. 2023, pp. 384–417. DOI: 10.1007/978-3-031-38545-2_13.

[11] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. "Security of Sanitizable Signatures Revisited". In: *PKC 2009*. Ed. by Stanislaw Jarecki and Gene Tsudik. Vol. 5443. LNCS. Springer, Berlin, Heidelberg, Mar. 2009, pp. 317–336. DOI: 10.1007/978-3-642-00468-1_18.

[12] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. "Unlinkability of Sanitizable Signatures". In: *PKC 2010*. Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. LNCS. Springer, Berlin, Heidelberg, May 2010, pp. 444–461. DOI: 10.1007/978-3-642-13013-7_26.

[13] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. "Sanitizable signatures: how to partially delegate control for authenticated data". In: *BIOSIG 2009*. Bonn: Gesellschaft für Informatik e.V., 2009, pp. 117–128. ISBN: 978-3-88579-249-1.

[14] Christina Brzuska, Henrich C. Pöhls, and Kai Samelin. "Non-interactive Public Accountability for Sanitizable Signatures". en. In: *Public Key Infrastructures, Services and Applications*. Ed. by Sabrina De Capitani di Vimercati and Chris Mitchell. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, 178–193. ISBN: 978-3-642-40012-4. DOI: 10.1007/978-3-642-40012-4_12.

[15] Xavier Bultel, Pascal Lafourcade, Russell W. F. Lai, Giulio Malavolta, Dominique Schröder, and Sri Aravinda Krishnan Thyagarajan. "Efficient Invisible and Unlinkable Sanitizable Signatures". In: *PKC 2019, Part I*. Ed. by Dongdai Lin and Kazue Sako. Vol. 11442. LNCS. Springer, Cham, Apr. 2019, pp. 159–189. DOI: 10.1007/978-3-030-17253-4_6.

[16] Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. "Chameleon-Hashes with Ephemeral Trap-

doors - And Applications to Invisible Sanitizable Signatures". In: *PKC 2017, Part II*. Ed. by Serge Fehr. Vol. 10175. LNCS. Springer, Berlin, Heidelberg, Mar. 2017, pp. 152–182. DOI: 10.1007/978-3-662-54388-7_6.

[17] Sébastien Canard and Amandine Jambert. "On Extended Sanitizable Signature Schemes". In: *CT-RSA 2010*. Ed. by Josef Pieprzyk. Vol. 5985. LNCS. Springer, Berlin, Heidelberg, Mar. 2010, pp. 179–194. DOI: 10.1007/978-3-642-11925-5_13.

[18] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. "Bonsai Trees, or How to Delegate a Lattice Basis". In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Berlin, Heidelberg, 2010, pp. 523–552. DOI: 10.1007/978-3-642-13190-5_27.

[19] Poulami Das, Andreas Erwig, Michael Meyer, and Patrick Struck. "Efficient Post-Quantum Secure Deterministic Threshold Wallets from Isogenies". In: *ASIACCS 24*. Ed. by Jianying Zhou, Tony Q. S. Quek, Debin Gao, and Alvaro A. Cárdenas. ACM Press, July 2024. DOI: 10.1145/3634737.3657008.

[20] David Derler, Kai Samelin, and Daniel Slamanig. "Bringing Order to Chaos: The Case of Collision-Resistant Chameleon-Hashes". In: *PKC 2020, Part I*. Ed. by Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas. Vol. 12110. LNCS. Springer, Cham, May 2020, pp. 462–492. DOI: 10.1007/978-3-030-45374-9_16.

[21] Muhammed F. Esgin, Ron Steinfeld, Dongxi Liu, and Sushmita Ruj. "Efficient Hybrid Exact/Relaxed Lattice Proofs and Applications to Rounding and VRFs". In: *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14085. Lecture Notes in Computer Science. Springer, 2023, pp. 484–517. DOI: 10.1007/978-3-031-38554-4_16. URL: https://doi.org/10.1007/978-3-031-38554-4_16.

[22] Marc Fischlin and Patrick Harasser. "Invisible Sanitizable Signatures and Public-Key Encryption are Equivalent". In: *ACNS 18International Conference on Applied Cryptography and Network Security*. Ed. by Bart Preneel and Frederik Vercauteren. Vol. 10892. LNCS. Springer, Cham, July 2018, pp. 202–220. DOI: 10.1007/978-3-319-93387-0_11.

[23] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. "Efficient Unlinkable Sanitizable Signatures from Signatures with Re-randomizable Keys". In: *PKC 2016, Part I*. Ed. by Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang. Vol. 9614. LNCS. Springer, Berlin, Heidelberg, Mar. 2016, pp. 301–330. DOI: 10.1007/978-3-662-49384-7_12.

[24] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. "Trapdoors for hard lattices and new cryptographic constructions". In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 197–206. DOI: 10.1145/1374376.1374407.

[25] Christian Janson and Patrick Struck. "Sponge-Based Authenticated Encryption: Security Against Quantum Attackers". In: *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022*. Ed. by Jung Hee Cheon and Thomas Johansson. Springer, Cham, Sept. 2022, pp. 230–259. DOI: 10.1007/978-3-031-17234-2_12.

[26] Shuichi Katsumata. "A New Simple Technique to Bootstrap Various Lattice Zero-Knowledge Proofs to QROM Secure NIZKs". In: *CRYPTO 2021, Part II*. Ed. by Tal Malkin and Chris Peikert. Vol. 12826. LNCS. Virtual Event: Springer, Cham, Aug. 2021, pp. 580–610. DOI: 10.1007/978-3-030-84245-1_20.

[27] Marek Klonowski and Anna Lauks. "Extended Sanitizable Signatures". In: *ICISC 06*. Ed. by Min Surp Rhee and Byoungcheon Lee. Vol. 4296. LNCS. Springer, Berlin, Heidelberg, 2006, pp. 343–355. DOI: 10.1007/11927587_28.

[28] Hugo Krawczyk and Tal Rabin. "Chameleon Signatures". In: *NDSS 2000*. The Internet Society, Feb. 2000.

[29] Stephan Krenn, Kai Samelin, and Dieter Sommer. "Stronger Security for Sanitizable Signatures". In: *DPM 2015*. Ed. by Joaquín García-Alfaro, Guillermo Navarro-Arribas, Alessandro Aldini, Fabio Martinelli, and Neeraj Suri. LNCS. Springer, 2015.

[30] Russell W. F. Lai, Tao Zhang, Sherman S. M. Chow, and Dominique Schröder. "Efficient Sanitizable Signatures Without Random Oracles". In: *ESORICS 2016, Part I*. Ed. by Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows. Vol. 9878. LNCS. Springer, Cham, Sept. 2016, pp. 363–380. DOI: 10.1007/978-3-319-45744-4_18.

[31] Yiming Li and Shengli Liu. "Tagged Chameleon Hash from Lattices and Application to Redactable Blockchain". In: *PKC 2024, Part II*. Ed. by Qiang Tang and Vanessa Teague. Vol. 14603. LNCS. Springer, Cham, Apr. 2024, pp. 288–320. DOI: 10.1007/978-3-031-57725-3_10.

[32] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. "Zero-Knowledge Arguments for Lattice-Based Accumulators: Logarithmic-Size Ring Signatures and Group Signatures Without Trapdoors". In: *EUROCRYPT 2016, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. LNCS. Springer, Berlin, Heidelberg, May 2016, pp. 1–31. DOI: 10.1007/978-3-662-49896-5_1.

[33] San Ling, Khoa Nguyen, Huaxiong Wang, and Yanhong Xu. "Constant-Size Group Signatures from Lattices". In: *PKC 2018, Part II*. Ed. by Michel Abdalla and Ricardo Dahab. Vol. 10770. LNCS. Springer, Cham, Mar. 2018, pp. 58–88. DOI: 10.1007/978-3-319-76581-5_3.

[34] Vadim Lyubashevsky and Ngoc Khanh Nguyen. "BLOOM: Bimodal Lattice One-out-of-Many Proofs and Applications". In: *ASIACRYPT 2022, Part IV*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13794. LNCS. Springer, Cham, Dec. 2022, pp. 95–125. DOI: 10.1007/978-3-031-22972-5_4.

[35] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon. "Lattice-Based Zero-Knowledge Proofs and Applications: Shorter, Simpler, and More General". In: *CRYPTO 2022, Part II*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13508. LNCS. Springer, Cham, Aug. 2022, pp. 71–101. DOI: 10.1007/978-3-031-15979-4_3.

[36] Daniele Micciancio and Chris Peikert. "Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller". In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Berlin, Heidelberg, Apr. 2012, pp. 700–718. DOI: 10.1007/978-3-642-29011-4_41.

[37] Daniele Micciancio and Oded Regev. "Worst-Case to Average-Case Reductions Based on Gaussian Measures". In: *45th FOCS*. IEEE Computer Society Press, Oct. 2004, pp. 372–381. DOI: 10.1109/FOCS.2004.72.

[38] *Module-Lattice-Based Digital Signature Standard*. National Institute of Standards and Technology, NIST FIPS PUB 204, U.S. Department of Commerce. Aug. 2024.

[39] Sunoo Park and Adam Sealfon. "It Wasn't Me! - Repudiability and Claimability of Ring Signatures". In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Cham, Aug. 2019, pp. 159–190. DOI: 10.1007/978-3-030-26954-8_6.

[40] Kai Samelin and Daniel Slamanig. "Policy-Based Sanitizable Signatures". In: *CT-RSA 2020*. Ed. by Stanislaw Jarecki. Vol. 12006. LNCS. Springer, Cham, Feb. 2020, pp. 538–563. DOI: 10.1007/978-3-030-40186-3_23.

[41] Adi Shamir and Yael Tauman. "Improved Online/Offline Signature Schemes". In: *CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. LNCS. Springer, Berlin, Heidelberg, Aug. 2001, pp. 355–367. DOI: 10.1007/3-540-44647-8_21.

# A Additional Preliminaries

This section introduces additional details relevant to understanding proofs in detail, but were left out of the main body due to their length.

## A.1 Digital Signature Scheme

A signature scheme is a public key based scheme to generate unforgeable signatures over arbitrary messages. Valid signatures can be generated only by a party with access to the secret key, while any party with access to the public key can validate signatures.

**Definition 50.** *A signature scheme* $\mathsf{Sig}$ *consists of three algorithms* $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$.

$(pk, sk) \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$: *This probabilistic key generation algorithm generates a key pair for the signature scheme. It takes the security parameter as input and outputs a key pair* $(pk, sk)$.

$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk, \mu)$: *This potentially probabilistic signing algorithm takes as input a secret key* $sk$ *and some message* $m$*, and outputs a signature* $\sigma$.

$b \leftarrow \mathsf{Vrfy}(pk, \sigma, \mu)$: *This deterministic algorithm takes as input a public key* $pk$*, a signature* $\sigma$ *and a message* $m$*, and outputs a boolean value of either* $1$ *or* $0$*, denoting a valid or invalid signature under the provided public key.*

**Definition 51.** *A signature scheme* $\mathsf{Sig}$ *is correct, iff for any security parameter* $\lambda \in \mathbb{N}$*, all generated key pairs* $(pk, sk) \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$*, all messages* $\mu \in \mathcal{M}$ *and all signatures* $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(sk, \mu)$*, it holds that* $\Pr[\mathsf{Vrfy}(pk, \sigma, \mu) = \top] = 1$.

## A.2 Public-key Encryption Scheme

A public-key encryption scheme is a public key based scheme which allows parties to securely communicate over an insecure channel. Messages can be encrypted under a public key and decrypted only by the party in possession of the corresponding secret key.

**Definition 52.** *A public key encryption scheme* $\Pi$ *consists of three algorithms:* $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$

$(pk, sk) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$: *This probabilistic key generation algorithm generates a key pair for the signature scheme. It takes no input and outputs a key pair* $(pk, sk)$.

$c \leftarrow\!\!\$\ \mathsf{Enc}(sk, \mu)$: *This potentially probabilistic encryption algorithm takes as input a secret key* $sk$ *and some message* $\mu$*, and outputs a ciphertext* $c$.

$\mu \leftarrow \mathsf{Dec}(pk, c)$: *This deterministic algorithm takes as input a public key* $pk$*, and a ciphertext* $c$*. It outputs either the decryption of the provided ciphertext under the given secret key or* $\perp$.

**Definition 53.** *A public-key encryption scheme* $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is correct, if* $\Pr[\mu' \neq \mu : (pk, sk) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda), c \leftarrow\!\!\$\ \mathsf{Enc}(pk, \mu), \mu' \leftarrow \mathsf{Dec}(sk, c)] = 0$.

### A.3 Chameleon Hash Function Uniqueness

While we claimed the construction in Section 3.1 does not have uniqueness, we did not yet define the security game. It should be hard for an adversary to find a message $\mu$ and two values $r, r'$ such that $\mu$ hashes to the same value with both $r$ and $r'$. The definition of the security game can be found in Fig. 12.

$\mathsf{CHUnique}_{\Pi,\mathcal{A}}(\lambda)$

$(\mathsf{pk}_{\mathsf{ch}}, h, \mu, r, r') \leftarrow\!\!\$\ \mathcal{A}(1^\lambda)$
**if** $\mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, h, \mu, r) = 1 \wedge \mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, h, \mu, r') = 1 \wedge r \neq r'$
  **return** 1
**return** 0

Fig. 12: Uniqueness security game.

**Definition 54.** *We say that a chameleon hash function* CH *has uniqueness, if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all qpt adversaries* $\mathcal{A}$ *we have that*

$$\Pr[\mathsf{CHUnique}_{\mathsf{CH},\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

### A.4 Additional Results concerning Lattices and Discrete Gaussians

To prove the security of our fully collision-resistant hash function of Section 3.1, we need some additional definitions, theorems and lemmas about lattices and discrete Gaussians.

**Theorem 55 ([37]).** *For any $n$-dimensional lattice $\Lambda$, vector $\mathbf{c} \in \mathbb{R}^n$, and reals $0 < \epsilon < 1, s > \eta_\epsilon(\Lambda)$, we have*

$$\Pr_{\mathbf{x} \leftarrow D_{\Lambda,s,\mathbf{c}}}[\|\mathbf{x} - \mathbf{c}\| > s\sqrt{n}] \leq \frac{1+\epsilon}{1-\epsilon} \cdot 2^{-n}\,.$$

Here, $\eta_\epsilon(\Lambda)$ describes the smoothing parameter of the lattice $\Lambda$ [37].

**Theorem 56 ([24]).** *Let $n$ and $q$ be positive integers with $q$ prime, and let $m \geq 2n \log q$. Then for all but a $2q^{-n}$ fraction of all $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and for any $s \geq \omega(\sqrt{\log m})$, the distribution of the syndrome $\mathbf{u} = \mathbf{Ae} \mod q$ is statistically close to uniform over $\mathbb{Z}_q^n$, where $\mathbf{e} \leftarrow D_{\mathbb{Z}^m,s}$.*

**Lemma 57 ([36]).** *For $\mathbf{R} \leftarrow D_{\mathbb{Z}^n,s}^m$ we know that there is a constant $C > 0$ such that for any $t \geq 0$ we have that $s_1(\mathbf{R}) \leq C \cdot s \cdot (\sqrt{n} + \sqrt{m} + t)$ with overwhelming probability.*

Heuristically, $C < 1$ so we ignore the factor.

### A.5 Labeled PKEs

The SSS construction in [6] requires the use of labeled PKEs to ensure strong invisibility. In Section 5.3, we presented a brief outline of labeled PKEs, how these are defined, and how one can generically transform (regular) PKEs to labeled PKEs such that the IND-CCA security of the former translates to the IND-CCA security of the latter.

Here, we explain more details about labeled PKEs, the definition of IND-CCA security of labeled PKEs, and we present the transform and the formal reduction from regular PKEs to labeled PKEs. The definitions are based on [6]. Intuitively, labels are circumstantial data, under which the encryption and decryption algorithms are run. While it is assumed in [6] that the label set can be unbounded, the particular use-case shows that the label will have a specific form, thus, imposing the restriction that the label set is the set of binary strings of a fixed length, which depends on the security level, causes no harm.

**Definition 58.** *A labeled PKE is a triple* $(\mathsf{KGen}_\tau, \mathsf{Enc}_\tau, \mathsf{Dec}_\tau)$ *such that*

$\mathsf{KGen}_\tau$ *Takes a security parameter* $\lambda$ *as input and creates key pairs* $(sk, pk)$

$\mathsf{Enc}_\tau$ *Takes a public key* $pk$*, a message* $\mu$*, and a label* $\tau$*, and outputs a ciphertext* $c$

$\mathsf{Dec}_\tau$ *Takes a secret key* $sk$*, a ciphertext* $c$*, and a label* $\tau$*, and outputs a message* $\mu$*.*

*The labeled PKE is (perfectly) correct, if for any key pair* $(sk, pk)$ *generated using the algorithm* $\mathsf{KGen}_\tau$*, any message* $\mu$ *and any label* $\tau$*, the equality*

$$\mathsf{Dec}_\tau(sk, \mathsf{Enc}_\tau(pk, \mu, \tau), \tau) = \mu$$

*holds. The labeled PKE is* $\epsilon$*-correct, if*

$$\mathbb{P}\left(\mathsf{Dec}_\tau(sk, \mathsf{Enc}_\tau(pk, \mu, \tau), \tau) = \mu\right) > \epsilon,$$

*where the probability is taken over the probability of the key generation, encryption, choices of the message, and choices of the label.*

The security notion that is interesting for the purpose of this work is IND-CCA security, which we define next.

**Definition 59.** *Let* $\Pi_\tau$ *be a labeled PKE. Then,* $\Pi_\tau$ *achieves* $\mathsf{IND\text{-}CCA}$ *security, if for all qpt adversaries* $\mathcal{A}$*, the advantage in winning the game* $\mathsf{IND\text{-}CCA}(\lambda)$ *as described in Fig. 13, which is given as*

$$\mathcal{A}_{\Pi_\tau}^{\mathsf{IND\text{-}CCA}}(\mathcal{A}),$$

*is negligible in the security parameter.*

$\underline{\mathsf{IND\text{-}CCA}_{\Pi_\tau,\mathcal{A}}(\lambda)}$

$(sk, pk) \leftarrow\!\!\$\ \mathsf{KGen}_\tau(\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}$
$(\mu_0, \mu_1, \tau^*, \mathsf{state}) \leftarrow\!\!\$\ \mathcal{A}_0^{\mathsf{Dec}\mathcal{O}_\tau}(pk)$
$c^* \leftarrow\!\!\$\ \mathsf{Enc}_\tau(pk, \mu_b, \tau^*)$
$b' \leftarrow\!\!\$\ \mathcal{A}_1^{\mathsf{Dec}\mathcal{O}'_\tau}(pk, c^*, \mathsf{state})$
**if** $b = b'$
  **return** 1
**return** 0

$\underline{\mathsf{Dec}\mathcal{O}_\tau(sk, \cdot, \cdot) \text{ with } (c, \tau)}$
**return** $\mathsf{Dec}_\tau(sk, c, \tau)$

$\underline{\mathsf{Dec}\mathcal{O}'_\tau(sk, \cdot, \cdot) \text{ with } (c, \tau)}$
**if** $c = c^*$
  **return** $\perp$
**return** $\mathsf{Dec}_\tau(sk, c, \tau)$

Fig. 13: IND-CCA security game for a labeled PKE $\Pi_\tau = (\mathsf{KGen}_\tau, \mathsf{Enc}_\tau, \mathsf{Dec}_\tau)$.

Labeled PKEs can be easily constructed from regular PKEs. For this, we assume that the label set is given as $\{0,1\}^n$ for some $n$ that depends on the security parameter.[7] Then, for any label $\tau$ and any message $\mu$, it is possible to recover both from the concatenation $\tau \| \mu$. This is the essential assumption on the behavior of the label used in the decryption in the following transform.

**Transform 60** *Given a regular PKE $\Pi = (\mathsf{KGen}, \mathsf{Dec}, \mathsf{Enc})$, the labeled PKE $\Pi_\tau = (\mathsf{KGen}_\tau, \mathsf{Enc}_\tau, \mathsf{Dec}_\tau)$ is defined as follows:*

$\mathsf{KGen}_\tau$ *The key generation is exactly* $\mathsf{KGen}$
$\mathsf{Enc}_\tau$ *On input $(pk, \mu, \tau)$, returns $\mathsf{Enc}(pk, \tau \| \mu)$*
$\mathsf{Dec}_\tau$ *On input $(sk, c, \tau)$, runs* $\mathsf{Enc}$ *with input $(sk, c)$ to receive $\tau' \| \mu'$. Then, checks whether $\tau = \tau'$ holds. If not, returns $\perp$, otherwise, it returns $\mu'$.*

**Proposition 61.** *Let $\Pi$ be a PKE and $\Pi_\tau$ the associated labeled PKE under Transform 60. For any adversary $\mathcal{A}$ against* IND-CCA *of $\Pi_\tau$, there exists an adversary $\mathcal{B}$ against* IND-CCA *of $\Pi$ such that*

$$\mathcal{A}^{\mathsf{IND\text{-}CCA}}_{\Pi_\tau}(\mathcal{A}) \leq \mathcal{A}^{\mathsf{IND\text{-}CCA}}_{\Pi}(\mathcal{B})\,.$$

*Proof.* The reduction $\mathcal{B}$ needs to simulate the two phases of the IND-CCA game of $\Pi_\tau$. First, $\mathcal{B}$ receives a public key as input, which it forwards to $\mathcal{A}$. On decryption queries by $\mathcal{A}$ with input $(c, \tau)$, where $c$ is a ciphertext and $\tau$ is a label, $\mathcal{B}$ sends $c$

[7] The definition in [15] allows labels to be bit strings of arbitrary length. For the given construction, however, labels will contain the public key which size depends on the security parameter.

to its own decryption oracle. When $\mathcal{B}$ receives $\tau' \| \mu'$ as response, it checks whether $\tau = \tau'$ holds. If so, $\mathcal{B}$ returns $\mu'$ to $\mathcal{A}$, otherwise $\perp$. The first phase finishes with $\mathcal{A}$ returning $(\mu_0, \mu_1, \tau^*, \mathsf{state})$ and $\mathcal{B}$ returns $(\tau^* \| \mu_0, \tau^* \| \mu_1, \mathsf{state} \| \tau^*)$. Note that the reduction explicitly has access to $\tau^*$ in the second phase.

The second phase begins with $\mathcal{B}$ receiving $c_b$, which—by the messages outputted after the first phase—is either a ciphertext for $\tau^* \| \mu_0$ or $\tau^* \| \mu_1$, and $\mathsf{state} \| \tau^*$. The reduction forwards $(c_b, \mathsf{state})$ to $\mathcal{A}$. Queries to the decryption oracle by $\mathcal{A}$ are of the form $(c, \tau)$. The reduction $\mathcal{B}$ forwards $c$ to its decryption oracle to receive $\tau' \| \mu'$ and returns $\mu'$ to $\mathcal{A}$, if $\tau' = \tau$. Note that $\mathcal{B}$ may receive $\perp$ from its decryption oracle. In this case, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. For $\mathcal{B}$ one inadmissible query is $c_b$ itself, while for $\mathcal{A}$, an inadmissible query is $(c_b, \tau^*)$. A priori, $\mathcal{A}$ can query $(c_b, \tau)$ for $\tau \neq \tau^*$. However, as the correct decryption of $c_b$ would yield $\tau^* \| \mu_b$, the transformed labeled PKE would return $\perp$, as the labels do not match. Thus, $\mathcal{B}$ can return $\perp$ whenever a query $(c_b, \cdot)$ is made by $\mathcal{A}$.

Finally, $\mathcal{A}$ returns a guess $b'$ and wins, if and only if $\mathsf{Dec}_\tau(sk, c_b) = \mu_{b'}$, or, in other words, $\mathsf{Dec}(sk, c_b) = \tau^* \| \mu_{b'}$. Hence, $\mathcal{B}$ returning $b'$ wins, if and only if $\mathcal{A}$ wins. □

# B Formal Details about the Security Notions and Correctness of SSS

This section will give formal definitions for all security notions as well as the relations between them.

## B.1 Security Notions

*Unforgeability* Intuitively, unforgeability requires that no adversary should be able to produce a valid message-signature pair that it has not seen before. Note that a valid output can either be signed directly or be the result of sanitizing another unrelated message. It is up to the attacker to break either the signing or the sanitizing part, correspondingly it gets access to both oracles. To exclude trivial wins, the game checks that the output message was not used in a query to either the signing or sanitizing oracle.

**Definition 62 (Unforgeability).** *A sanitizable signature scheme* SSS *is (strongly) unforgeable, if for all qpt adversaries $\mathcal{A}$, the advantage in winning the game $\mathsf{Exp}^{\mathsf{unf}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$ as described in Fig. 14, defined as*

$$\mathsf{Adv}^{\mathrm{unf}}_{\mathsf{SSS},\mathcal{A}}(\lambda) \coloneqq \Pr\left[\mathsf{Exp}^{\mathsf{unf}}_{\mathsf{SSS},\mathcal{A}}(\lambda) = 1\right]$$

*is negligible in the security parameter $\lambda$.*

$\mathsf{Exp}^{\mathsf{unf}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$

$(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}}) \leftarrow\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$

$(pk_{\mathsf{San}}, sk_{\mathsf{San}}) \leftarrow\!\$\ \mathsf{KGen}_{\mathsf{San}}(1^\lambda)$

$\mathcal{Q}_{\mathsf{Sig}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{San}} \leftarrow \emptyset$

$(\mu^*, \sigma^*) \leftarrow\!\$\ \mathcal{A}^{\mathsf{Sign}\mathcal{O},\mathsf{Sanit}\mathcal{O},\mathsf{Proof}\mathcal{O}}(pk_{\mathsf{Sig}}, pk_{\mathsf{San}})$

**if** $\mathsf{Vrfy}(\mu^*, \sigma^*, pk_{\mathsf{Sig}}, pk_{\mathsf{San}}) = \top$

$\wedge\ (\mu^*, *, pk_{\mathsf{San}}, \sigma^*) \notin \mathcal{Q}_{\mathsf{Sig}}$

$\wedge\ (\mu^*, *, \sigma^*, pk_{\mathsf{Sig}}) \notin \mathcal{Q}_{\mathsf{San}}$

**return** 1 **else return** 0

$\mathsf{Sign}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot)$ with $(\mu, \mathrm{ADM}, pk_{\mathsf{San}})$

$\sigma \leftarrow \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$

$\mathcal{Q}_{\mathsf{Sig}} \leftarrow \mathcal{Q}_{\mathsf{Sig}} \cup (\mu, \mathrm{ADM}, pk_{\mathsf{San}}, \sigma)$

**return** $\sigma$

$\mathsf{Sanit}\mathcal{O}(sk_{\mathsf{San}}, \cdot, \cdot, \cdot, \cdot)$ with $(\mu, \mathrm{MOD}, \sigma, pk_{\mathsf{Sig}})$

$(\mu', \sigma') \leftarrow\!\$\ \mathsf{Sanit}(\mu, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk_{\mathsf{Sig}})$

$\mathcal{Q}_{\mathsf{San}} \leftarrow \mathcal{Q}_{\mathsf{San}} \cup \{(\mu, \mu', \sigma', pk_{\mathsf{Sig}})\}$

**return** $(\mu', \sigma')$

$\mathsf{Proof}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot, \cdot)$ with $(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, pk_{\mathsf{San}})$

$\pi \leftarrow\!\$\ \mathsf{Proof}(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, sk_{\mathsf{Sig}}, pk_{\mathsf{San}})$

**return** $\pi$

Fig. 14: Game based security definition for (strong) unforgeability. Inclusion of the gray box yields strong unforgeability.

*(weak) Immutability* Intuitively, a malicious sanitizer must not be able to modify blocks that are not admissible for modification. The adversary is only given the public key of a signer and has to output a sanitizer public key (which it can generate honestly or maliciously) and a valid message signature pair. There are two options for the adversary to succeed: Either it has never queried the signing oracle with the sanitizer public key it has output. This would mean that there should not exist any valid signatures which verify under the output public key. The other option is to output a message that none of its queries to the signing oracle can be sanitized to. In other words, it has output an inadmissible modification, which still verifies.

A scheme achieves weak immutability if the adversary is not allowed to query the Sign oracle under a message that is admissable to be modified to its output, but a different sanitizer key.

**Definition 63 (Immutability).** *A sanitizable signature scheme* $\mathsf{SSS}$ *is (weakly) immutable, if for all qpt adversaries* $\mathcal{A}$*, the advantage in winning the game* $\mathsf{Exp}^{\mathsf{imm}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$ *as described in Fig. 15, defined as*

$$\mathsf{Adv}^{\mathrm{imm}}_{\mathsf{SSS},\mathcal{A}}(\lambda) \coloneqq \Pr\left[\mathsf{Exp}^{\mathsf{imm}}_{\mathsf{SSS},\mathcal{A}}(\lambda) = 1\right]$$

*is negligible in the security parameter* $\lambda$*.*

*Privacy* Intuitively, no adversary should be able to recover any information about the content of the original message before it was sanitized. The adversary is given access to a sign oracle, a sanitation oracle and a proof oracle (as the proof might leak information about the signer secret key) and a left-or-right oracle. Before

$\mathsf{Exp}^{(\mathsf{w})\mathsf{imm}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$

$(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$
$\mathcal{Q}_{\mathsf{Sig}} \leftarrow \emptyset$
$(pk^*_{\mathsf{San}}, \mu^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Sign}\mathcal{O},\mathsf{Proof}\mathcal{O}}(pk_{\mathsf{Sig}})$
**if** $\mathsf{Vrfy}(\mu^*, \sigma^*, pk_{\mathsf{Sig}}, pk^*_{\mathsf{San}}) = \top$
  $\wedge\ (\forall(\mu_i, \mathrm{ADM}_i, pk_{\mathsf{San}}, *) \in \mathcal{Q}_{\mathsf{Sig}}:$
    $\mu^* \notin \{\mathrm{MOD}(\mu_i) \mid \mathrm{MOD}(\mathrm{ADM}_i) = \top\}$
    $\vee\, pk^*_{\mathsf{San}} \neq pk_{\mathsf{San}})$
  **return** 1
**return** 0

$\mathsf{Sign}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot)$ with $(\mu, \mathrm{ADM}, pk_{\mathsf{San}})$

$\sigma \leftarrow \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$
$\mathcal{Q}_{\mathsf{Sig}} \leftarrow \mathcal{Q}_{\mathsf{Sig}} \cup (\mu, \mathrm{ADM}, pk_{\mathsf{San}}, \sigma)$
**return** $\sigma$

$\mathsf{Proof}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot, \cdot)$
  with $(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, pk_{\mathsf{San}})$

$\pi \leftarrow\!\!\$\ \mathsf{Proof}(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, sk_{\mathsf{Sig}}, pk_{\mathsf{San}})$
**return** $\pi$

Fig. 15: Game based security definition for (weak) immutability. Omission of the gray box yields weak immutability.

making its decision, the adversary may observe an arbitrary amount of signing or sanitizing queries under the challenge public keys. The adversary wins if it can decide which of the two input messages are sanitized by the left-or-right oracle.

**Definition 64 (Privacy).** *A sanitizable signature scheme* $\mathsf{SSS}$ *is private, if for all qpt adversaries* $\mathcal{A}$*, the advantage in winning the game* $\mathsf{Exp}^{\mathsf{prv}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$ *as described in Fig. 16, defined as*

$$\mathsf{Adv}^{\mathrm{prv}}_{\mathsf{SSS},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{prv}}_{\mathsf{SSS},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2}\right|$$

*is negligible in the security parameter* $\lambda$*.*

*Unlinkability* Intuitively, no adversary given a sanitized message-signature pair should be able to identify the original message-signature pair that has been sanitized. In other words, it must be infeasible to link the given sanitized document to the original signed document. As a consequence, two different sanitized messages cannot be identified as belonging to the same initial message. This is a strictly stronger requirement than privacy and also modeled using a similar left-or-right oracle, but with more control for the adversary. Unlike in the privacy game, the unlinkability adversary can see the signatures before sanitation occurs. The adversary can provide two sets of message-signature pairs with identical admissible blocks (because the scheme might not be invisible) as well as modification instructions that map them to the same resulting message. Depending on the secret bit $b$, the oracle will sanitize and return either the left or the right message. To succeed, the adversary must correctly determine which of the input messages is being sanitized by the oracle.

In the literature, first introduced in [15], there exists a weaker notion of unlinkability, which restricts any adversary to only sanitizing messages it has

$\mathsf{Exp}^{\mathsf{prv}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$

$(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$

$(pk_{\mathsf{San}}, sk_{\mathsf{San}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{San}}(1^\lambda)$

$b \leftarrow\!\!\$\ \{0,1\}, \mathcal{Q}_{\mathsf{Sig}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{San}} \leftarrow \emptyset$

$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Sign}\mathcal{O},\mathsf{Sanit}\mathcal{O},\mathsf{Proof}\mathcal{O},\mathsf{LoRSanit}_{\mathsf{prv}}\mathcal{O}}(pk_{\mathsf{Sig}}, pk_{\mathsf{San}})$

**return** $b = b'$

$\mathsf{Sign}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot)$ with $(\mu, \mathrm{ADM}, pk_{\mathsf{San}})$

$\sigma \leftarrow \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$

$\mathcal{Q}_{\mathsf{Sig}} \leftarrow \mathcal{Q}_{\mathsf{Sig}} \cup (\mu, \mathrm{ADM}, pk_{\mathsf{San}}, \sigma)$

**return** $\sigma$

$\mathsf{Sanit}\mathcal{O}(sk_{\mathsf{San}}, \cdot, \cdot, \cdot, \cdot)$ with $(\mu, \mathrm{MOD}, \sigma, pk_{\mathsf{Sig}})$

$(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{Sanit}(\mu, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk_{\mathsf{Sig}})$

$\mathcal{Q}_{\mathsf{San}} \leftarrow \mathcal{Q}_{\mathsf{San}} \cup \{(\mu, \mu', \sigma', pk_{\mathsf{Sig}})\}$

**return** $(\mu', \sigma')$

$\mathsf{Proof}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot, \cdot)$ with $(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, pk_{\mathsf{San}})$

$\pi \leftarrow\!\!\$\ \mathsf{Proof}(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, sk_{\mathsf{Sig}}, pk_{\mathsf{San}})$

**return** $\pi$

$\mathsf{LoRSanit}_{\mathsf{prv}}\mathcal{O}(sk_{\mathsf{Sig}}, sk_{\mathsf{San}}, b, \cdot, \cdot)$ with $((\mu_0, \mathrm{MOD}_0), (\mu_1, \mathrm{MOD}_1), \mathrm{ADM})$

$\sigma_b \leftarrow\!\!\$\ \mathsf{Sign}(\mu_b, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$

$(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{Sanit}(\mu_b, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk_{\mathsf{Sig}})$

**if** $\mathrm{MOD}_0(\mu_0) = \mathrm{MOD}_1(\mu_1)$

  **return** $(\mu', \sigma')$

**return** $\bot$

Fig. 16: Game based security definition for privacy.

previously honestly generated using the sign oracle. We capture this by the optional check in the $\mathsf{LoRSanit}_{\mathsf{unl}}\mathcal{O}$ oracle.

**Definition 65 (Unlinkability).** *A sanitizable signature scheme* SSS *is (weakly) unlinkable, if for all qpt adversaries* $\mathcal{A}$*, the advantage in winning the game* $\mathsf{Exp}^{(\mathrm{w})\mathsf{unl}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$ *as described in Fig. 17, defined as*

$$\mathsf{Adv}^{(\mathrm{w})\mathrm{unl}}_{\mathsf{SSS},\mathcal{A}}(\lambda) \coloneqq \left| \Pr\left[\mathsf{Exp}^{(\mathrm{w})\mathsf{unl}}_{\mathsf{SSS},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2} \right|$$

*is negligible in the security parameter* $\lambda$*.*

*Transparency* Intuitively, no outside adversary should be able to determine the creator of a signature. This property is modeled using a left-or-right oracle, that either sanitizes or signs the result of the modification as a new message. The adversary is successful, if the can determine if the message-signature pairs it is receiving are output from the signing or the sanitizing algorithm. Transparency is mutually exclusive with non-interactive public accountability, as they described opposite requirements for the signatures. This notion of transparency is also frequently referred to as "proof-restricted transparency", to highlight the fact that the proof oracle cannot be misused for trivial wins. This is relevant in cases where the proof oracle works without using the additional message-signature pairs, i.e., it is “history free”. If the proof is not history free, the adversary cannot use the proof oracle to trivially win the game, as it does not know learn the original

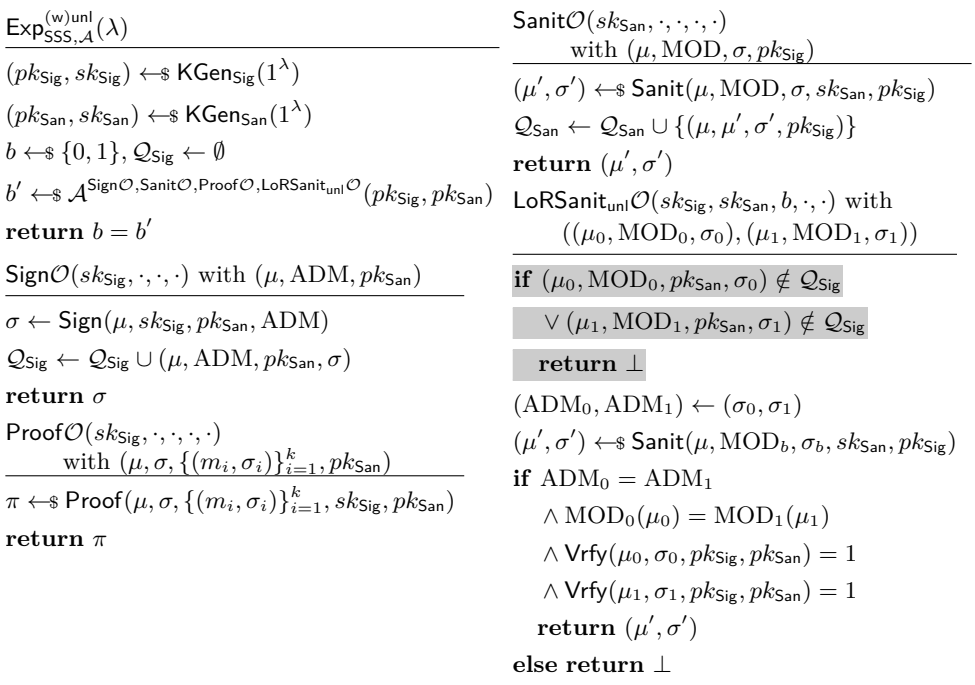


Fig. 17: Game based security definition for (weak) unlinkability. Inclusion of the gray box yields the weak variant.

signatures from the $\mathsf{Sanit}/\mathsf{Sign}\mathcal{O}$ oracle which it would have to provide to the proof oracle. Consequently, construction like [13], which are not history free, can achieve the stronger non proof-restricted notion of transparency. This was first observed in [12], For a detailed breakdown of the evolution of the transparency property we refer to [29].

Many works [16, 6, 40] have defaulted to the proof-restricted variant, meaning they do not consider the non proof-restricted variant at all and are calling the proof-restricted variant "transparency". We will follow this approach in this paper and use the term "transparency" to refer to proof-restricted transparency. Note that some works like [30, 15] make this history-freeness explicit by disallowing the input of additional message-signature pairs into the proof oracle, making the proof-restricted variant the only viable definition. There also exists a strong variant of transparency. The term was first introduced by [3], but their description was later formalized by [16] to become the formal notion of invisibility. For strong transparency, the definition by [29] has gained the most traction. For strong transparency, the proof-restriction is slightly weakened, allowing queries where the message, but not the signature, has been output by the $\mathsf{Sanit}/\mathsf{Sign}\mathcal{O}$ oracle. Interestingly, some works like [16, 15, 6] are using this stronger definition as their regular version of transparency, while others like [12, 23] are using the

weaker variant as their regular version of transparency without discussing the strengthened notion.

**Definition 66 (Proof Restricted Transparency).** *A sanitizable signature scheme* $\mathsf{SSS}$ *is (strongly)proof restricted transparent (from now own we only say transparency), if for all qpt adversaries* $\mathcal{A}$*, the advantage in winning the game* $\mathsf{Exp}^{\mathsf{(s)transp}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$ *as described in Fig. 18, defined as*

$$\mathsf{Adv}^{\mathrm{(s)transp}}_{\mathsf{SSS},\mathcal{A}}(\lambda) \coloneqq \left|\Pr\left[\mathsf{Exp}^{\mathsf{(s)transp}}_{\mathsf{SSS},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2}\right|$$

*is negligible in the security parameter* $\lambda$*.*

$\mathsf{Exp}^{\mathsf{(s)transp}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$

$(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$

$(pk_{\mathsf{San}}, sk_{\mathsf{San}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{San}}(1^\lambda)$

$b \leftarrow\!\!\$\ \{0,1\}, \mathcal{Q}_{\mathsf{Sig}} \leftarrow \emptyset$

$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Sign}\mathcal{O},\mathsf{Sanit}\mathcal{O},\mathsf{Proof}'\mathcal{O},\mathsf{Sanit/Sign}\mathcal{O}}(pk_{\mathsf{Sig}}, pk_{\mathsf{San}})$

**return** $b = b'$

$\mathsf{Sign}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot)$ with $(\mu, \mathrm{ADM}, pk_{\mathsf{San}})$

$\sigma \leftarrow \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$

$\mathcal{Q}_{\mathsf{Sig}} \leftarrow \mathcal{Q}_{\mathsf{Sig}} \cup (\mu, \mathrm{ADM}, pk_{\mathsf{San}}, \sigma)$

**return** $\sigma$

$\mathsf{Sanit}\mathcal{O}(sk_{\mathsf{San}}, \cdot, \cdot, \cdot, \cdot)$ with $(\mu, \mathrm{MOD}, \sigma, pk_{\mathsf{Sig}})$

$(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{Sanit}(\mu, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk_{\mathsf{Sig}})$

$\mathcal{Q}_{\mathsf{San}} \leftarrow \mathcal{Q}_{\mathsf{San}} \cup \{(\mu, \mu', \sigma', pk_{\mathsf{Sig}})\}$

**return** $(\mu', \sigma')$

$\mathsf{Proof}'\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot, \cdot)$ with $(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, pk'_{\mathsf{San}})$

**if** $pk'_{\mathsf{San}} = pk_{\mathsf{San}} \wedge$
$((\mu, \sigma) \in \mathcal{Q}_{\mathsf{sigsan}} \vee \exists i : (\mu_i, \sigma_i) \in \mathcal{Q}_{\mathsf{sigsan}})$
**return** $\perp$

$\pi \leftarrow\!\!\$\ \mathsf{Proof}(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, sk_{\mathsf{Sig}}, pk_{\mathsf{San}})$

**return** $\pi$

$\mathsf{Sanit/Sign}\mathcal{O}(sk_{\mathsf{Sig}}, sk_{\mathsf{San}}, pk_{\mathsf{Sig}}, pk_{\mathsf{San}}, b, \cdot, \cdot, \cdot)$ with $(\mu, \mathrm{MOD}, \mathrm{ADM})$

**if** $\mathrm{MOD}(\mathrm{ADM}) = \perp$ **return** $\perp$

$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$

$(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{Sanit}(\mu, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk_{\mathsf{Sig}})$

**if** $b = 0$
$\mathcal{Q}_{\mathsf{sigsan}} \leftarrow \mathcal{Q}_{\mathsf{sigsan}} \cup (\mu', \sigma')$
**return** $(\mu', \sigma')$

**else**
$\sigma'' \leftarrow \mathsf{Sign}(\mu', sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$
$\mathcal{Q}_{\mathsf{sigsan}} \leftarrow \mathcal{Q}_{\mathsf{sigsan}} \cup (\mu', \sigma'')$
**return** $(\mu', \sigma'')$

Fig. 18: Game based security definition for (strong) transparency. Inclusion of the gray box yields strong transparency.

*Invisibility* Intuitively, no efficient adversary should be able to detect whether any specific message block is admissible for modification or not.

To formalize this notion, a left-or-right signing oracle $\mathsf{LoRAdm}\mathcal{O}$ is employed in combination with a restricted sanitizing oracle $\mathsf{Sanit}'\mathcal{O}$. The sanitizing oracle is modified to prevent trivial wins. The adversary is limited to sanitizing

message-signature pairs it previously acquired from the left-or-right oracle and to introducing modifications which are allowed in both $\mathrm{ADM}_0$ and $\mathrm{ADM}_1$ to prevent trivial wins. or to outputs of the modified sanitizer oracle $\mathsf{Sanit}'\mathcal{O}$. The left-or-right oracle behaves like a signing oracle, but takes two sets of admissible blocks $\mathrm{ADM}_0$ and $\mathrm{ADM}_1$ as input. Depending on the secret bit $b$ sampled by the experiment, the message is signed with either the left or the right set of admissible blocks. Additionally, access to $\mathsf{Proof}\mathcal{O}$ is provided to allow the adversary to attempt to learn $sk_{\mathsf{Sig}}$ The adversary wins, if it correctly guesses if the oracle is using $\mathrm{ADM}_0$ or $\mathrm{ADM}_1$ to sign the given messages.

A list $\mathcal{Q}$ is used to track previous queries of the adversary and to evaluate if a given query is allowed or not.

**Definition 67 (Invisibility).** *A sanitizable signature scheme* $\mathsf{SSS}$ *is invisible, if for all qpt adversaries* $\mathcal{A}$*, the advantage in winning the game* $\mathsf{Exp}^{\mathsf{inv}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$ *as described in Fig. 19, defined as*

$$\mathsf{Adv}^{\mathrm{inv}}_{\mathsf{SSS},\mathcal{A}}(\lambda) \coloneqq \left|\Pr\left[\mathsf{Exp}^{\mathsf{inv}}_{\mathsf{SSS},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2}\right|$$

*is negligible in the security parameter* $\lambda$*.*

$\mathsf{Exp}^{\mathsf{inv}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$

$(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$
$(pk_{\mathsf{San}}, sk_{\mathsf{San}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{San}}(1^\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}, \mathcal{Q} \leftarrow \emptyset$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Sanit}'\mathcal{O},\mathsf{Proof}\mathcal{O},\mathsf{LoRAdm}\mathcal{O}}(pk_{\mathsf{Sig}}, pk_{\mathsf{San}})$
**return** $b = b'$

$\mathsf{Sanit}'\mathcal{O}(sk_{\mathsf{San}}, pk_{\mathsf{Sig}}, \cdot, \cdot, \cdot)$ with $(\mu, \mathrm{MOD}, \sigma)$

**if** $\nexists(\mu, \sigma, \mathrm{ADM}) \in \mathcal{Q} : \mathrm{MOD}(\mathrm{ADM}) = \top$
  **return** $\bot$
$(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{Sanit}(\mu, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk_{\mathsf{Sig}})$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mu', \sigma', \mathrm{ADM}\}$
**return** $(\mu', \sigma)$

$\mathsf{Proof}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot, \cdot)$
  with $(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, pk_{\mathsf{San}})$

$\pi \leftarrow\!\!\$\ \mathsf{Proof}(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, sk_{\mathsf{Sig}}, pk_{\mathsf{San}})$
**return** $\pi$

$\mathsf{LoRAdm}\mathcal{O}(sk_{\mathsf{Sig}}, b, \cdot, \cdot, \cdot)$ with
  $(\mu, \mathrm{ADM}_0, \mathrm{ADM}_1)$

$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{MOD}_b)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mu, \sigma, \mathrm{ADM}_0 \cap \mathrm{ADM}_1\}$
**return** $\sigma$

Fig. 19: Game based security definition for invisibility.

**Definition 68 (Strong Invisibility).** *A sanitizable signature scheme* $\mathsf{SSS}$ *is strongly invisible, if for all qpt adversaries* $\mathcal{A}$*, the advantage in winning the game* $\mathsf{Exp}^{\mathsf{inv}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$ *as described in Fig. 20, defined as*

$$\mathsf{Adv}^{\mathrm{inv}}_{\mathsf{SSS},\mathcal{A}}(\lambda) \coloneqq \left|\Pr\left[\mathsf{Exp}^{\mathsf{inv}}_{\mathsf{SSS},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2}\right|$$

*is negligible in the security parameter $\lambda$.*

$\mathsf{Exp}^{\mathsf{s-inv}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$

$(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$
$(pk_{\mathsf{San}}, sk_{\mathsf{San}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{San}}(1^\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}, \mathcal{Q} \leftarrow \emptyset$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Proof}\mathcal{O},\mathsf{LoRAdm}'\mathcal{O},\mathsf{Sanit}''\mathcal{O}}(pk_{\mathsf{Sig}}, pk_{\mathsf{San}})$
**return** $b = b'$

$\mathsf{Sanit}''\mathcal{O}(sk_{\mathsf{San}}, \cdot, \cdot, \cdot, \cdot)$ with $(pk'_{\mathsf{Sig}}, \mu, \mathrm{MOD}, \sigma)$

**if** $pk'_{\mathsf{Sig}} = pk_{\mathsf{Sig}} \wedge \nexists(\mu, \sigma, \mathrm{ADM}) \in \mathcal{Q} : \mathrm{MOD}(\mathrm{ADM}) = \top$
  **return** $\bot$
$(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{Sanit}(\mu, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk'_{\mathsf{Sig}})$
**if** $pk'_{\mathsf{Sig}} = pk_{\mathsf{Sig}} \wedge \exists(\mu, \sigma, \mathrm{ADM}') \in \mathcal{Q} : \mathrm{MOD}(\mathrm{ADM}') = \top$
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mu', \sigma', \mathrm{ADM}\}$
**return** $(\mu', \sigma)$

$\mathsf{LoRAdm}'\mathcal{O}(sk_{\mathsf{Sig}}, b, \cdot, \cdot, \cdot, \cdot)$ with
  $(pk'_{\mathsf{San}}, \mu, \mathrm{ADM}_0, \mathrm{ADM}_1)$

**if** $pk'_{\mathsf{Sig}} = pk_{\mathsf{Sig}} \wedge \mathrm{ADM}_0 = \mathrm{ADM}_1$
  **return** $\bot$
$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk'_{\mathsf{San}}, \mathrm{MOD}_b)$
**if** $pk'_{\mathsf{Sig}} = pk_{\mathsf{Sig}} \wedge \exists(\mu, \sigma, \mathrm{ADM}') \in \mathcal{Q} : \mathrm{MOD}(\mathrm{ADM}') = \top$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mu, \sigma, \mathrm{ADM}_0 \cap \mathrm{ADM}_1\}$
**return** $\sigma$

$\mathsf{Proof}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot, \cdot)$
  with $(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, pk_{\mathsf{San}})$

$\pi \leftarrow\!\!\$\ \mathsf{Proof}(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, sk_{\mathsf{Sig}}, pk_{\mathsf{San}})$
**return** $\pi$

Fig. 20: Game based security definition for strong invisibility.

Strong invisibility strengthens this notions by allowing an adversary to query its oracles under arbitrary signer public keys, simulating a dishonest signer. In this case the aforementioned restrictions for the left or right and sanitize oracle only apply when queried with the challenge public key. Note that in both the regular and the strong variant of invisibility, a signing oracle can be simulated by querying the corresponding left or right oracle with $\mathrm{ADM}_0 = \mathrm{ADM}_1$.

*Accountability* Intuitively, accountability formalizes the ability to hold both signer and sanitizer responsible for messages they have signed. The strong accountability

notion extends this to the signature, holding the respective party responsible for the signatures they have created. For technical reasons this notion is split into signer-accountability and sanitizer accountability. Both notions formalize the inability of the respective party to frame the other party for a signature they have not produced; when both notions are achieved one has accountability.

For signer-accountability we refer to Definition 37 in the main body of the paper.

**Definition 69 (Sanitizer-Accountability).** *A sanitizable signature scheme* $\mathsf{SSS}$ *is (strongly) sanitizer-accountable, if for all qpt adversaries* $\mathcal{A}$*, the advantage in winning the game* $\mathsf{Exp}^{\text{(s)san-acc}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$ *as described in Fig. 21, defined as*

$$\mathsf{Adv}^{\text{(s)sig-acc}}_{\mathsf{SSS},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Exp}^{\text{(s)sig-acc}}_{\mathsf{SSS},\mathcal{A}}(\lambda) = 1\right]$$

*is negligible in the security parameter* $\lambda$*.*

$\mathsf{Exp}^{\text{(s)san-acc}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$

$(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$
$\mathcal{Q}_{\mathsf{Sig}} \leftarrow \emptyset$
$(pk^*_{\mathsf{San}}, \mu^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Sign}\mathcal{O},\mathsf{Proof}\mathcal{O}}(pk_{\mathsf{Sig}})$
$\pi \leftarrow \mathsf{Proof}(sk_{\mathsf{Sig}}, \mu^*, \sigma^*, \{\mu_i, \sigma_i :$
$\quad (\mu_i, *, \sigma_i, *) \in \mathcal{Q}_{\mathsf{Sig}}\}, pk_{\mathsf{San}})$
**if** $(\mu^*, *, pk^*_{\mathsf{San}}, \sigma^*) \notin \mathcal{Q}_{\mathsf{Sig}}$
$\quad \wedge\ \mathsf{Vrfy}(\mu^*, \sigma^*, pk^*_{\mathsf{Sig}}, pk_{\mathsf{San}}) = \top$
$\quad \wedge\ \mathsf{Judge}(\mu^*, \sigma^*, pk^*_{\mathsf{Sig}}, pk_{\mathsf{San}}, \pi) = \mathsf{Sig}$
$\quad$ **return** 1
**else return** 0

$\mathsf{Sign}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot)$ with $(\mu, \mathrm{ADM}, pk_{\mathsf{San}})$

$\sigma \leftarrow \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$
$\mathcal{Q}_{\mathsf{Sig}} \leftarrow \mathcal{Q}_{\mathsf{Sig}} \cup (\mu, \mathrm{ADM}, pk_{\mathsf{San}}, \sigma)$
**return** $\sigma$

$\mathsf{Proof}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot, \cdot)$
$\quad$ with $(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, pk_{\mathsf{San}})$

$\pi \leftarrow\!\!\$\ \mathsf{Proof}(\mu, \sigma, \{(m_i, \sigma_i)\}_{i=1}^k, sk_{\mathsf{Sig}}, pk_{\mathsf{San}})$
**return** $\pi$

Fig. 21: Game based security definition for (strong) sanitizer accountability. Inclusion of the gray box yields strong sanitizer accountability.

**Definition 70 (Accountability).** *A sanitizable signature scheme* $\mathsf{SSS}$ *is (strongly) accountable, if it is both (strongly) signer-accountable and (strongly) sanitizer-accountable.*

*Non-Interactive Public Accountability* Intuitively, non-interactive public accountability requires that for any signature an external observer can reliably decide whether this signature was created by the signer or the sanitizer without any additional information. This is formalized by removing the ability to use the proof algorithm (which takes $sk_{\mathsf{Sig}}$ as input). Consequently, the $\mathsf{Judge}$ algorithm receives the symbol $\perp$ as input instead of the proof $\pi$. If the $\mathsf{Judge}$ algorithm still works as expected the scheme is non-interactively publicly accountable.

**Definition 71 (Non-Interactive Public Accountability).** *A sanitizable signature scheme* $\mathsf{SSS}$ *achieves non-interactive public accountability, if for all qpt adversaries* $\mathcal{A}$*, the advantage in winning the game* $\mathsf{Exp}^{\mathsf{nipa}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$ *as described in Fig. 22, defined as*

$$\mathsf{Adv}^{\mathrm{nipa}}_{\mathsf{SSS},\mathcal{A}}(\lambda) \coloneqq \left|\Pr\left[\mathsf{Exp}^{\mathsf{nipa}}_{\mathsf{SSS},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2}\right|$$

*is negligible in the security parameter* $\lambda$*.*

$\mathsf{Exp}^{\mathsf{nipa}}_{\mathsf{SSS},\mathcal{A}}(\lambda)$

$(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$
$(pk_{\mathsf{San}}, sk_{\mathsf{San}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{San}}(1^\lambda)$
$\mathcal{Q}_{\mathsf{Sig}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{San}} \leftarrow \emptyset$
$(pk^*, \mu^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Sign}\mathcal{O},\mathsf{Sanit}\mathcal{O}}(pk_{\mathsf{Sig}}, pk_{\mathsf{San}})$
**if** $(\mu^*, *, pk_{\mathsf{San}}, \sigma) \notin \mathcal{Q}_{\mathsf{Sig}}$
  $\wedge\ \mathsf{Vrfy}(\mu^*, \sigma^*, pk^*_{\mathsf{Sig}}, pk^*) = \top$
  $\wedge\ \mathsf{Judge}(\mu^*, \sigma^*, pk^*_{\mathsf{Sig}}, pk^*, \bot) = \mathsf{Sig}$
  **return** 1
**elseif** $(*, \mu^*, \sigma^*, pk^*_{\mathsf{Sig}}) \notin \mathcal{Q}_{\mathsf{San}}$
  $\wedge\ \mathsf{Vrfy}(\mu^*, \sigma^*, pk^*, pk_{\mathsf{San}}) = \top$
  $\wedge\ \mathsf{Judge}(\mu^*, \sigma^*, pk^*, pk_{\mathsf{San}}, \bot) = \mathsf{San}$
  **return** 1
**else return** 0

$\mathsf{Sign}\mathcal{O}(sk_{\mathsf{Sig}}, \cdot, \cdot, \cdot)$ with $(\mu, \mathrm{ADM}, pk_{\mathsf{San}})$

$\sigma \leftarrow \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$
$\mathcal{Q}_{\mathsf{Sig}} \leftarrow \mathcal{Q}_{\mathsf{Sig}} \cup (\mu, \mathrm{ADM}, pk_{\mathsf{San}}, \sigma)$
**return** $\sigma$

$\mathsf{Sanit}\mathcal{O}(sk_{\mathsf{San}}, \cdot, \cdot, \cdot, \cdot)$
  with $(\mu, \mathrm{MOD}, \sigma, pk_{\mathsf{Sig}})$

$(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{Sanit}(\mu, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk_{\mathsf{Sig}})$
$\mathcal{Q}_{\mathsf{San}} \leftarrow \mathcal{Q}_{\mathsf{San}} \cup \{(\mu, \mu', \sigma', pk_{\mathsf{Sig}})\}$
**return** $(\mu', \sigma')$

Fig. 22: Game based security definition for non-interactive public accountability.

**Definition 72.** *A sanitizable signature scheme* $\mathsf{SSS}$ *is correct, if signing correctness, sanitizing correctness, and proof correctness hold.*

*Signing Correctness. For all security parameter* $\lambda \in \mathbb{N}$*, any key pair* $(sk_{\mathsf{Sig}}, pk_{\mathsf{Sig}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$*, any message* $\mu \in \mathcal{M}$*, any valid* ADM*, and for every* $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$*, it holds that*

$$\Pr[\mathsf{Vrfy}(\mu, \sigma, pk_{\mathsf{Sig}}, pk_{\mathsf{San}}) = \top] = 1.$$

*Sanitizing Correctness. For all security parameter* $\lambda \in \mathbb{N}$*, any key pair* $(sk_{\mathsf{Sig}}, pk_{\mathsf{Sig}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$*, any message* $\mu \in \mathcal{M}$*, any valid* ADM*, and for every* $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$*, any valid* MOD *with* $\mathrm{MOD}(\mathrm{ADM}) = \top$*, and any sanitized message with signature* $(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{Sanit}(\mu, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk_{\mathsf{Sig}})$*, it holds that*

$$\Pr[\mathsf{Vrfy}(\mu', \sigma', pk_{\mathsf{Sig}}, pk_{\mathsf{San}}) = \top] = 1.$$

*Proof Correctness. For all security parameters* $\lambda \in \mathbb{N}$*, any key pair* $(sk_{\mathsf{Sig}}, pk_{\mathsf{Sig}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{Sig}}(1^\lambda)$*, any message* $\mu \in \mathcal{M}$*, any valid* ADM*, any signature* $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mu, sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mathrm{ADM})$ *with* $\mathsf{Vrfy}(\mu, \sigma, pk_{\mathsf{Sig}}, pk_{\mathsf{San}}) = \top$*, any valid* MOD *that does not violate* ADM*, any sanitized message-signature pair* $(\mu', \sigma') \leftarrow\!\!\$\ \mathsf{Sanit}(\mu, \mathrm{MOD}, \sigma, sk_{\mathsf{San}}, pk_{\mathsf{Sig}})$*, with* $\mathsf{Vrfy}(\mu', \sigma', pk_{\mathsf{Sig}}, pk_{\mathsf{San}}) = \top$*, and polynomial many additional signature-message pairs* $(\mu_i, \sigma_i)$ *for* $1 \leq i \leq k$*, and for all proofs* $\pi \leftarrow\!\!\$\ \mathsf{Proof}(\mu, \sigma, \{(\mu_i, \sigma_i)\}_{i=1}^k, sk_{\mathsf{Sig}}, pk_{\mathsf{Sig}}, pk_{\mathsf{San}})$ *and* $\pi' \leftarrow\!\!\$\ \mathsf{Proof}(\mu', \sigma', \{(\mu_i, \sigma_i)\}_{i=1}^k, sk_{\mathsf{Sig}}, pk_{\mathsf{Sig}}, pk_{\mathsf{San}})$ *it holds that*

$$\begin{aligned} &\Pr[\mathsf{Sig} \leftarrow \mathsf{Judge}(\mu, \sigma, pk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \pi)] = 1 \\ \wedge\, &\Pr[\mathsf{San} \leftarrow \mathsf{Judge}(\mu', \sigma', pk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \pi')] = 1. \end{aligned}$$

### B.2 Relations Between Security Notions

In Fig. 23 we provide an overview of the relations between the different security properties of sanitizable signature schemes and observe an additional implication between non-interactive public accountability and accountability in Theorem 73.

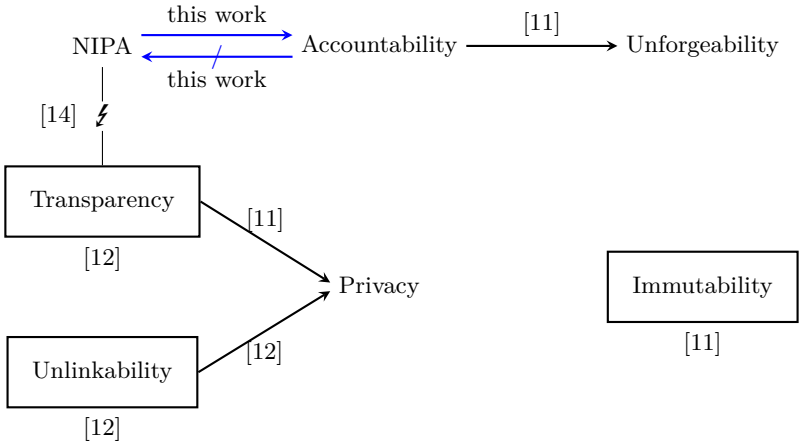


Fig. 23: The relations between different security notions for SSS. Arrows indicate implications, a lightning indicates two notions are conflicting. A box indicates that the notion is independent of all other notions. Citations next to the implications or boxes cite the source where this was first observed.

In [12] it is stated, that accountability is independent of all other security properties. At the time non-interactive public accountability has not yet been established as a security notion for sanitizable signature schemes. Nowadays this claim no longer holds due to the following relation between accountability and non-interactive public accountability.

**Theorem 73.** *A sanitizable signature scheme* SSS *that is non-interactively publicly accountable is also accountable.*

*Proof.* We prove Theorem 73 using reductions. We make use of the fact that, per definition, the proof algorithm of a non-interactively publicly accountable SSS only outputs $\perp$. Note that accountability is a compound notion consisting of signer and sanitizer accountability. Hence, we need to provide two reductions: one breaking non-interactive public accountability given an adversary $\mathcal{A}_{\text{sig-acc}}$ against signer accountability, and one given an adversary $\mathcal{A}_{\text{san-acc}}$ against sanitizer accountability that both succeed with non-negligible probability.

First, we assume $\mathcal{A}_{\text{san-acc}}$ as described above and construct a reduction $\mathcal{B}_{\mathsf{San}}$ that breaks non-interactive public accountability with non-negligible probability. The reduction $\mathcal{B}_{\mathsf{San}}$ receives two public keys $(pk_{\mathsf{Sig}}, pk_{\mathsf{San}})$ as part of the experiment and will forward $pk_{\mathsf{Sig}}$ to $\mathcal{A}_{\text{san-acc}}$. Oracle queries by $\mathcal{A}_{\text{san-acc}}$ to $\mathsf{Proof}\mathcal{O}$ can always be answered with $\perp$ per definition, and queries to $\mathsf{Sign}\mathcal{O}$ are forwarded to the reductions own sign oracle. Finally, $\mathcal{A}_{\text{san-acc}}$ will output $(pk^*_{\mathsf{San}}, m^*, \sigma^*)$ where $\mathsf{Vrfy}(m^*, \sigma^*, pk_{\mathsf{Sig}}, pk^*_{\mathsf{San}}) = \top$. Lastly, $\mathcal{B}_{\mathsf{San}}$ forwards the adversary's output as its output. The reduction $\mathcal{B}_{\mathsf{San}}$ is efficient, perfectly simulates the sanitizer accountability game for $\mathcal{A}_{\mathsf{San}}$, and has the same non-negligible success probability as $\mathcal{A}_{\mathsf{San}}$ by definition, which we assume to be non-negligible.

The reduction $\mathcal{B}_{\mathsf{Sig}}$ works analogously with two slight modifications: It does not need to simulate the proof oracle and only forwards the output $(pk^*_{\mathsf{Sig}}, m^*, \sigma^*)$, omitting $\pi^*$.

We have constructed two reductions, $\mathcal{B}_{\mathsf{San}}$ and $\mathcal{B}_{\mathsf{Sig}}$, and have shown that for any SSS it holds that $\mathsf{Adv}^{\mathsf{nipa}}_{\mathsf{SSS},\mathcal{A}_{\text{sig-acc}}}(\lambda) \leq \mathsf{Adv}^{\text{sig-acc}}_{\mathsf{SSS},\mathcal{B}_{\mathsf{Sig}}}(\lambda)$ and $\mathsf{Adv}^{\mathsf{nipa}}_{\mathsf{SSS},\mathcal{A}_{\text{san-acc}}}(\lambda) \leq \mathsf{Adv}^{\text{san-acc}}_{\mathsf{SSS},\mathcal{B}_{\mathsf{San}}}(\lambda)$, which proves the result. □

**Theorem 74.** *A sanitizable signature scheme* SSS *that is accountable is not necessarily non-interactively publicly accountable.*

*Proof (Sketch).* Assume any accountable SSS scheme, where the accountability is realized by the judge algorithm evaluating its input $\pi$. This scheme is clearly not non-interactively publicly accountable, as in this setting, the judge will receive only the symbol $\perp$ instead of the proof $\pi$. □

## C Additional Information on BFF$^+$09

In this appendix, we provide additional information on the BFF$^+$09 construction by Brzuska et al. [11]. We begin with the notion of tagged chameleon hashes, a corrected security notion for collision-resistance, and a transform to build such fully collision-resistant tagged chameleon hash functions from fully collision-resistant chameleon hash functions (without tags), as constructed in Fig. 7. Afterwards, we present the details on the construction.

### C.1 Fully Collision-Resistant Tagged Chameleon Hash Functions

The construction by Brzuska et al. [11] requires a tagged chameleon hash function $\mathsf{CH}$. A tag chameleon hash function takes a tag as additional input to the hashing algorithm. The adapt procedure takes a message, tag, and randomness, and an additional second message as input. It outputs a new tag and a new randomness, such that the hash values of the input triple coincides with the hash of the second input message and the output tag-randomness pair.

As the collision-resistance notion shown in Fig. 10 is unachievable, we present a new formalization which follows closely the f-CR game in Fig. 2. For full collision-resistance, the adversary receives receives a public key $pk$ and access to an adapt oracle which internally chooses uniformly random tags. On input $\mu$, the oracle that outputs $(\mathrm{TAG}, r)$ stores the information $(h, \mathrm{TAG}, \mu)$, where $h$ is the hash of $(\mathrm{TAG}, \mu, r)$. If the adversary outputs a hash collision by providing two tags $(\mathrm{TAG}_1, \mathrm{TAG}_2)$, two messages $(\mu_1, \mu_2)$, and two randomnesses $(r_1, r_2)$ with $(\mathrm{TAG}_1, \mu_1) \neq (\mathrm{TAG}_2, \mu_2)$. Let $h_i = \mathsf{CH}(pk, \mathrm{TAG}_i, \mu_i, r_i)$. Then, in the full collision-resistance game, the output is accepted, if the one of the triples of $(h_1, \mathrm{TAG}_1, \mu_1)$ and $(h_2, \mathrm{TAG}_2, \mu_2)$ has not been part of an adapt query. This description closely resembles the f-CR game in Fig. 2. We note that the security notion for tagged chameleon hash functions here is not related immediately to tagged chameleon hash function security in [31].

Finally, it is easy to see that f-CR tagged chameleon hashes can be constructed from f-CR chameleon hash functions without tags. Indeed, given a chameleon hash function $\mathsf{CH} = (\mathsf{CKGen}, \mathsf{CHash}, \mathsf{CHashCheck}, \mathsf{CAdapt})$, we construct the tagged version $(\mathsf{CKGen}_{\mathrm{TAG}}, \mathsf{CHash}_{\mathrm{TAG}}, \mathsf{CHashCheck}_{\mathrm{TAG}}, \mathsf{CAdapt}_{\mathrm{TAG}})$ as follows:

$\mathsf{CKGen}_{\mathrm{TAG}} = \mathsf{CKGen}$
$\mathsf{CHash}_{\mathrm{TAG}}$ takes $(\mathrm{TAG}, \mu, r)$ and outputs $\mathsf{CHash}(\mathrm{TAG} \| \mu, r)$
$\mathsf{CHashCheck}_{\mathrm{TAG}}$ takes $(h, \mathrm{TAG}, \mu, r)$ and outputs $\mathsf{CHashCheck}(h, \mathrm{TAG} \| \mu, r)$
$\mathsf{CAdapt}_{\mathrm{TAG}}$ takes $(h, \mathrm{TAG}, \mu, r, \mu')$, samples $\mathrm{TAG}'$ uniformly randomly, and outputs $(\mathrm{TAG}', \mathsf{CAdapt}(h, \mathrm{TAG} \| \mu, r, \mathrm{TAG}' \| \mu))$.

With standard techniques it can be shown that f-CR of the tagged chameleon hash reduces to f-CR of the underlying chameleon hash function.

### C.2 The BFF$^+$09 Construction

In this section, we introduce the construction by Brzuska et al. [11], which we have analyzed more closely in Section 5.5.

**Construction 75** *Let* PRG *be a pseudorandom generator,* PRF *a pseudorandom function,* $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *a signature scheme, and* CH *a fully collision-resistant tagged chameleon hash. A sanitizable signature scheme* $\mathsf{SanSig} = (\mathsf{KGen}_{\mathsf{Sig}}, \mathsf{KGen}_{\mathsf{San}}, \mathsf{Sign}, \mathsf{Sanit}, \mathsf{Vrfy}, \mathsf{Proof}, \mathsf{Judge})$ *is defined as follows:*

**Key Generation.** *The signer key* $(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}})$ *is generated using* KGen. *The sanitizer key* $(pk_{\mathsf{San}}, sk_{\mathsf{San}})$ *is generated using* CKGen.

**Signing.** *The algorithm* $\mathsf{Sign}$*, on input* $\mu \in \{0,1\}^{tl}$*,* $sk_{\mathsf{Sig}}$*,* $pk_{\mathsf{San}}$*, and* ADM*, picks* $\text{NONCE} \leftarrow\!\!\$\ \{0,1\}^n$ *at random, computes* $x \leftarrow\!\!\$\ \mathsf{PRF}(\kappa, \text{NONCE})$*,* $\text{TAG} \leftarrow\!\!\$\ \mathsf{PRG}(x)$*, and picks* $r[j]$ *for each* $j \in \text{ADM}$ *at random. It sets*

$$h[j] \leftarrow\!\!\$\ \mathsf{CH}(pk_{\mathsf{San}}, \text{TAG}, (j, \mu_j, pk_{\mathsf{Sig}}); r[j]),$$

*if* $j \in \text{ADM}$*, or* $\mu_j$ *else. Then, it computes* $\sigma_0 \leftarrow \mathsf{Sign}(sk_{\mathsf{Sig}}, (h, pk_{\mathsf{San}}, \text{ADM}))$ *where* $h = (h[1], \ldots, h[l])$ *and returns*

$$\sigma \leftarrow (\sigma_0, \text{TAG}, \text{NONCE}, \text{ADM}, r[j_1], \ldots, r[j_k])$$

*where* $j_i \in \text{ADM}$*.*

**Sanitizing.** *The algorithm* $\mathsf{Sanit}$ *on input a message* $\mu$*, information* MOD*, a signature* $\sigma = (\sigma_0, \text{TAG}, \text{NONCE}, \text{ADM}, r[j_1], \ldots, r[j_k])$*,* $pk_{\mathsf{Sig}}$*, and* $sk_{\mathsf{San}}$*, first checks that each modification in* MOD *is admissible according to* ADM *and that* $\sigma_0$ *is a valid signature fo* $(h, pk_{\mathsf{San}}, \text{ADM})$*. If not, it outputs* $\perp$*. Otherwise, for each* $j \in \text{ADM}$*, it lets* $\mu'[j]$ *be the modified block of* $\mu_j$*, possibly* $\mu'[j] = \mu_j$*, picks* $\text{NONCE}'$ *and* $\text{TAG}'$ *randomly and replaces* $r[j]$ *for each admissible* $j$ *by* $r'[j] = \mathsf{CAdapt}(sk_{\mathsf{San}}, \text{TAG}, (j, \mu_j, pk_{\mathsf{Sig}}), r[j], \text{TAG}', (j, \mu'[j], pk_{\mathsf{Sig}}))$*. It outputs* $\mu'$ *and* $\sigma' = (\sigma_0, \text{TAG}', \text{NONCE}', \text{ADM}, r'[j_1], \ldots, r'[j_k])$*.*

**Verification.** *The algorithm* $\mathsf{Vrfy}$ *on input a message* $\mu \in \{0,1\}^{tl}$ *and a signature*

$$\sigma \leftarrow (\sigma_0, \text{TAG}, \text{NONCE}, \text{ADM}, r[i_1], \ldots, r[i_k]),$$

$pk_{\mathsf{Sig}}$*, and* $pk_{\mathsf{San}}$*, computes the hashes* $h[j]$ *as above, sets* $h = (h[1], \ldots, h[l])$ *and runs* $\Sigma.\mathsf{Vrfy}(pk_{\mathsf{Sig}}, (h, pk_{\mathsf{San}}, \text{ADM}), \sigma_0)$*.*

**Proof.** *The algorithm* $\mathsf{Proof}$ *on input* $sk_{\mathsf{Sig}}$*,* $\mu$*,* $\sigma$*, a sequence* $(\mu_i, \sigma_i)$*, and* $pk_{\mathsf{San}}$*, searches for* $(\text{TAG}_i, (j, \mu_i[j], pk_{\mathsf{Sig}}), r_i[j])$ *such that*

$$\mathsf{CH}(pk_{\mathsf{San}}, \text{TAG}_i, (j, \mu_i[j], pk_{\mathsf{Sig}}), r_i[j]) = \mathsf{CH}(pk_{\mathsf{San}}, \text{TAG}, (j, \mu_j, pk_{\mathsf{Sig}}), r[j]),$$

*for some distinct* $(\text{TAG}, (j, \mu_j, pk_{\mathsf{Sig}}), r[j])$ *in* $\mu$*,* $\sigma$*, where* $\text{TAG}_i = \mathsf{PRG}(x_i)$ *for* $x_i = \mathsf{PRF}(\kappa, \text{NONCE}_i)$ *for the value* $\text{NONCE}_i$ *in* $\sigma_i$*. If such data is found, it returns*

$$\pi \leftarrow (\text{TAG}_i, (j, \mu_i[j], pk_{\mathsf{Sig}}), r_i[j], x_i),$$

*else it outputs* $\perp$*.*

**Judge.** *The algorithm* $\mathsf{Judge}$*, on input* $\mu$*,* $\sigma$*,* $pk_{\mathsf{Sig}}$*,* $pk_{\mathsf{San}}$*, and*

$$\pi = (\text{TAG}_\pi, (j, \mu_\pi[j], pk_{\mathsf{Sig}}), r_\pi[j], x_\pi)$$

*checks if*

$$\begin{aligned} & pk_{\mathsf{Sig}} = pk_{\mathsf{Sig},\pi}, \\ \wedge\ & \mathsf{CH}(pk_{\mathsf{San}}, \text{TAG}, (j, \mu_j, pk_{\mathsf{Sig}}), r[j]) \\ & \quad = \mathsf{CH}(pk_{\mathsf{San}}, \text{TAG}_\pi, (j, \mu_\pi[j], pk_{\mathsf{Sig}}), r_\pi[j]), \\ \wedge\ & \textit{block } j \textit{ is admissible}, \\ \wedge\ & \text{TAG}_\pi = \mathsf{PRG}(x_\pi). \end{aligned}$$

*If all conditions are satisfied,* $\mathsf{Judge}$ *outputs* $\mathsf{San}$*, otherwise, it outputs* $\mathsf{Sig}$*.*

## D Lattice-Based Instantiations from Standard Primitives

Following the different classes of constructions, we start with a discussion of those instantiations that rely solely on standard primitives. This includes the FH18 construction by Fischlin and Harasser [22], followed by the BFLS09 construction by Brzuska et al. [13].

*The* FH18 *Construction.* Fischlin and Harasser [22] show in their paper that public-key encryption is equivalent to invisible sanitizable signature schemes. They give a construction that relies solely on standard primitives: public-key encryption schemes and digital signature schemes.

We only give a the high-level idea of their construction and refer to [22] for the details.

The construction works as follows. The signer key pair consists of a key pair for the underlying signature scheme $\Sigma$. The sanitizer key pair also consists of a key pair of the underlying signature scheme $\Sigma$ and additionally contains a key pair of the public-key encryption scheme $\Pi$. To sign a message $\mu$, it is chopped into blocks $\mu_1, \ldots, \mu_l$ which are then signed individually using freshly generated key pairs of $\Sigma$, yielding signatures $\sigma_1, \ldots, \sigma_l$. The message is chopped into blocks and each block is signed with a freshly generated key pair. For each block, the signer encrypts either the corresponding secret key—if the block is sanitizable—or the zero-string—if the block is fixed. This enables the sanitizer to obtain the secret key for each to, which allows to change the message and sign it with that secret key. Finally, the signer (or the sanitizer, if it sanitized a message) uses its secret key to sign the message alongside the individual signatures and ciphertexts.

**Theorem 76.** *If the signature scheme $\Sigma$ is instantiated with one of the candidates of the set $\{\mathsf{Dilithium}, \mathsf{Falcon}\}$, and the public-key encryption scheme instantiated with* Kyber.CPA*, then the* FH18 *construction [22] is an unforgeable, immutable, private, publicly accountable, and invisible sanitizable signature scheme against qpt adversaries.*

*Proof.* The theorem follows straightforwardly from [22]. Their proof directly transfers to *qpt* adversaries, as none of the oracles are available in superposition and there is no random oracle. Furthermore, the adversaries in their reductions are black box, therefore there is no rewinding or cloning taking place, which would also contradict *qpt* security. □

The above theorem shows that sanitizable signatures are possible from lattices and, in fact, any assumptions that allow the construction of both public-key encryption and signature schemes.[8] We note, however, that [22] is a more theoretical work. Their main result shows that sanitizable signatures and public-key encryption are equivalent. The construction is fairly inefficient—signatures are very large since they contain a public key for each message block.

[8] This precludes hash-based cryptography which only allows for signatures but not public-key encryption

*The* BFLS09 *Construction.* A more efficient construction is the BFLS09 construction by Brzuska et al. [13]. The construction works as follows. The key pairs for signer and sanitizer consists of a key pair for the underlying signature scheme $\Sigma$. To sign a message $\mu$, it is chopped into blocks $\mu_1, \ldots, \mu_l$ and the signer creates two signatures. First, $\sigma_{fix}$, which essentially signs all immutable message blocks. Second, $\sigma_{full}$, which signs the entire message—immutable and admissible blocks. In order to sanitize a signature, the sanitizer changes the message $\mu$ to $\mu^*$ and replaces $\sigma_{full}$ by signing the $\mu^*$. Verification works by verifying that $\sigma_{fix}$ is a valid signature under the public key of the signer and that $\sigma_{full}$ is a valid signature under the public key of either signer or sanitizer.

The signature $\sigma_{fix}$ on the immutable blocks prevents the sanitizer to change any immutable blocks, as these are always verified against the public key of the signer. The simplicity of the construction comes at the expense of not achieving transparency: whether $\sigma_{full}$ verifies against the public key of the signer or the sanitizer immediately reveals if a message was sanitized or not. On the other hand, the construction is minimal in the sense that it relies solely on signature schemes—allowing easy instantiation from, say, Dilithium.

Below we state that the BFLS09 construction can be instantiated from lattices, which follows straightforward from [13].

**Theorem 77.** *If the signature scheme $\Sigma$ is instantiated with one of the candidates of the set* {Dilithium, Falcon}*, then the* BFLS09 *construction [13] is an unforgeable, immutable, private, and accountable sanitizable signature scheme against qpt adversaries.*

## E Lattice-Based Instantiations from Advanced Signatures

In this section, we cover the different SSS constructions that rely on various types of advanced signature schemes (and not on chameleon hash functions). These include the following constructions: the FKM$^+$16 construction [23], the BFLS10 construction [12], the LZCS16−2 construction [30], and the BLL$^+$19 construction [15].

*The* FKM$^+$16 *Construction.* The construction by Fleischhacker et al. [23] relies on signature schemes with rerandomizable keys. On a high-level, these are signature schemes that allow to randomize a key pair $(sk, pk)$, using some randomness $r$, into a new key pair $(sk', pk')$ such that (1) $(sk', pk')$ is a matching key pair, i.e., signatures generated using $sk'$ verify under $pk'$, and (2) re-randomized keys are indistinguishable from keys generated via KGen.

Alkeilani Alkadri et al. [2] showed that rerandomizable signature schemes can be constructed from lattices. However, they consider a weaker notion, where only the re-randomized public key should be hard to distinguish from a public key obtained via KGen.[9] In fact, [2] identifies obstacles due to usage of Gaussian distributions for lattice-based constructions which typically result in different

[9] This modification is motivated via their goal of showing public keys to be unlinkable.

distributions for keys generated via KGen and re-randomized ones. Thus, the construction from [2] cannot be used to instantiate the construction by Fleischhacker et al. [23]. This leaves the open problem of constructing rerandomizable signature schemes according to [23] from lattice[10] assumptions.

*The* BFLS10 *Construction.* The construction by Brzuska et al. [12] uses group signatures and is historically the first sanitizable signature scheme that does makes use of advanced signature notions: In this work, the construction is based on group signatures, which are required to satisfy *anonymity*, *traceability*, and *non-frameability*.

The construction can be instantiated using the group signature that has been developed by Ling et al. [33].

**Theorem 78.** *The* BFLS10 *construction [12] instantiated with a signature from* {*Dilithium*, *Falcon*} *and the group signature from [33] is a lattice-based sanitizable signature scheme that satisfies unforgeabilty, immutability, signer- and sanitizer accountability, transparency, privacy, and unlinkability.*

*The* LZCS16−2 *Construction.* In [30], Lai et al. give two distinct constructions. One deploys chameleon hash functions, which we discuss later. Here, we focus on the other construction, which uses accountable ring signatures as its primary component in combination with deterministic signature schemes. For both, we have lattice based primitives available, which achieve the required security properties and can be used for a generic instantiation. For deterministic signatures ML-DSA [38] can be used in the deterministic mode, as standardized by NIST. For accountable ring signatures, Beullens et al. [8] provide a lattice based as well as an isogeny based instantiation, both of which are suitable for instantiating this construction in practice.

*The* BLL$^+$19 *Construction.* Finally, we review the construction by Bultel et al. [15], which already played a crucial role in the instantiations of SSS constructions using chameleon hash functions. There, we used the transform described in [15] to increase the security to include unforgeability and accountability, if their security proofs in the respective constructions required the chameleon hash to satisfy uniqueness—the property that our lattice-based CHF fails to achieve.

In [15], so-called equivalence class signatures (EQS) have been used to create a weak SSS. Then, requiring (strongly) unforgeable VRS, it is shown how to add unforgeability and accountability features to the weak SSS resulting in a secure SSS. The equivalence class signatures in [15] rely on pairing problems which are not post-quantum secure. Further investigations regarding the weak SSS construction may provide useful insights on generating other weak sanitizable signatures. However, the pairing problem has to be replaced to ensure post-quantum security. On the other hand, we make extensive use of the transform in Theorem 38.

[10] Recently, Das et al. [19] showed a construction from isogenies.

# F Full Proofs for Theorem Statements

## F.1 Learning with Rounding-Based LITs

In Lemma 18 we stated that there exists an LWR-based LIT. We now prove this lemma by constructing and proving secure such a LIT. First, we need to define learning with rounding and its rounding function. We define the rounding function $\lfloor\cdot\rfloor_p : \mathbb{Z}_q \to \mathbb{Z}_p, x \mapsto \lfloor (p/q)\cdot x \rfloor$. If we use that function on vectors, we use it component-wise.

**Definition 79 (Learning with Rounding).** *In the learning with rounding (LWR) problem* $\mathsf{LWR}_{n,q,p,\chi}$*, the challenger picks a secret* $\mathbf{s} \in \mathbb{Z}_q^n$ *from a distribution* $\chi$ *and defines the LWR distribution that samples a uniform* $\mathbf{a} \leftarrow\!\!\$\ \mathbb{Z}_q^n$ *and outputs* $(\mathbf{a}, \lfloor \mathbf{s}^t\mathbf{a}\rfloor_p)$*. Then, a qpt adversary* $\mathcal{A}$ *can ask for samples from either the LWR distribution or from the uniform distribution over* $\mathbb{Z}_q^{n+1}$ *and has to decide from what distribution the samples are.*

*In the search version of LWR* $\mathsf{sLWR}_{n,q,p,\chi}$ *the qpt adversary gets samples from the LWR distribution and has to find* $\mathbf{s}$*.*

Furthermore, we need a lemma telling us when an LWR secret is unique, similar to [9, Lemma 1] for module LWE. For that, define $B_a^b = \{\mathbf{x} \in \mathbb{Z}_q^b : \|\mathbf{x}\|_\infty \leq a\}$.

**Lemma 80.** *Let* $n, q, p, \beta > 0$ *with* $q > p$ *and* $0 < \beta < q$*. Let* $q = \prod_{i=1}^d q_i^{e_i}$*, where* $q_i$ *are prime and let* $j$ *be such that* $q_j = \min\{q_i\}$*. Choose* $m > 0$ *with* $m \in \mathsf{poly}(n)$ *such that* $\left(\frac{2\lfloor q/p\rfloor}{q_j}\right)^{m/n} \cdot 2\beta$ *is smaller than some constant* $c < 1$*. Then, it holds that*

$$\Pr[\exists \mathbf{s}, \hat{\mathbf{s}} \in B_\beta^n : \mathbf{s} \neq \hat{\mathbf{s}}, \lfloor \mathbf{s}^t\mathbf{A}\rfloor_p = \lfloor \hat{\mathbf{s}}^t\mathbf{A}\rfloor_p] \leq \left(\frac{q_j^{e_j-1}}{q}\right)^m \cdot (2\beta)^n \cdot (\lfloor q/p\rfloor)^m,$$

*which is negligible in* $n$*.*

*Proof.* Assume that we have $\mathbf{s}, \hat{\mathbf{s}} \in \mathbb{Z}_q^n$ with $\mathbf{s} \neq \hat{\mathbf{s}}$ and $\lfloor \mathbf{s}^t\mathbf{A}\rfloor_p = \lfloor \hat{\mathbf{s}}^t\mathbf{A}\rfloor_p$. We can interpret this LWR equation as an LWE equation with fixed errors. Since the rounding function divides $\mathbb{Z}_q$ into $p$ intervals of size $\lfloor q/p\rfloor$ and maps an element from $\mathbb{Z}_q$ to the index of the interval, we know that by scaling this index by $\lfloor q/p\rfloor$ we get the rounded representation of $x$ in $\mathbb{Z}_q$. Furthermore, since elements in the interval $[k\cdot\lfloor q/p\rfloor, (k+1)\cdot\lfloor q/p\rfloor - 1]$ are mapped to $k$ and thus scaled to $k\lfloor q/p\rfloor$, we know that the difference between $x$ and its rounded representation in $\mathbb{Z}_q$ differ by at most $\lfloor q/p\rfloor$. Thus, there exist $\mathbf{e}, \hat{\mathbf{e}} \in \mathbb{Z}_q^m$ with $\|\mathbf{e}\|_\infty, \|\hat{\mathbf{e}}\|_\infty \leq \lfloor q/p\rfloor$ such that $\mathbf{s}^t\mathbf{A} + \mathbf{e} = \lfloor \mathbf{s}^t\mathbf{A}\rfloor_p \cdot \lfloor q/p\rfloor \mod q$ and $\hat{\mathbf{s}}^t\mathbf{A} + \hat{\mathbf{e}} = \lfloor \hat{\mathbf{s}}^t\mathbf{A}\rfloor_p \cdot \lfloor q/p\rfloor \mod q$. Then, we know that

$$\begin{aligned}
&(\mathbf{s}^t - \hat{\mathbf{s}}^t)\mathbf{A} + \mathbf{e} - \hat{\mathbf{e}} = \mathbf{0}\\
\Leftrightarrow &(\mathbf{s}^t - \hat{\mathbf{s}}^t)\mathbf{A} = -\mathbf{e} + \hat{\mathbf{e}}\\
\Leftrightarrow &\bar{\mathbf{s}}^t\mathbf{A} = \bar{\mathbf{e}}
\end{aligned}$$

for some $\bar{\mathbf{s}} \in B_{2\beta}^n$ and some $\bar{\mathbf{e}} \in B_{2\lfloor q/p \rfloor}^m$. Thus, we can apply [9, Theorem 6] to say that such $\bar{\mathbf{s}}, \bar{\mathbf{e}}$ exist only with probability

$$\begin{aligned}\epsilon(n) &:= \Pr[\exists(\bar{\mathbf{s}}, \bar{\mathbf{e}}) \in B_{2\beta}^n \times B_{2\lfloor q/p \rfloor}^m : \bar{\mathbf{s}}^t \mathbf{A} = \bar{\mathbf{e}}^t; \mathbf{A} \hookleftarrow \$ \, \mathbb{Z}_q^{n \times m}] \\ &\leq \left( \frac{1 + z_q^{\max}}{q} \right)^m \cdot (2\beta)^n \cdot (2\lfloor q/p \rfloor)^m,\end{aligned}$$

where $z_w^{\max} := \max\{|\mathcal{Z}_{w,s}| : s \in \mathbb{Z}_w \backslash \{0\}\}$ with $\mathcal{Z}_{w,s} := \{a \in \mathbb{Z}_w \backslash \{0\} : a \cdot s = 0\}$. Since $q = \prod_{i=1}^d q_i^{e_i}$, we know that $\mathbb{Z}_q \cong \mathbb{Z}_{q_1}^{e_1} \times \ldots \times \mathbb{Z}_{q_d}^{e_d}$. Furthermore, we know that $z_{q_i^{e_i}}^{\max} = q_i^{e_i - 1} - 1$ for all $i$, since there are $q_i^{e_i - 1} - 1$ non-zero multiples of $q_i$ that are smaller than $q_i^{e_i}$ and $q_i^{e_i - 1}$ times any other multiple is zero. Therefore, an element $(0, \ldots, 0, q_j^{e_j - 1}, 0, \ldots, 0)$, where the index $j$ is such that $q_j^{e_j} = \min_i\{q_i^{e_i}\}$, maximises $z_q^{\max}$. Thus, we have

$$z_{\max} = -1 + q_j^{e_j - 1} \prod_{i \neq j}^{d} q_i = \frac{q}{q_j} - 1$$

Using this and the definition of $m$ on the bound for $\epsilon(n)$, we get

$$\begin{aligned}\epsilon(n) &\leq \frac{1}{q_j^m} \cdot (2\beta)^n \cdot (2\lfloor q/p \rfloor)^m \\ &\leq \left( \left( \frac{2\lfloor q/p \rfloor}{q_j} \right)^{m/n} \cdot 2\beta \right)^n \\ &\leq c^n,\end{aligned}$$

which is negligible in $n$ since $c < 1$. □

**Corollary 81.** *For $q$ prime, $n, p, \beta > 0$ with $p < q$ and $0 < \beta < q$. Choose $m$ such that $4\beta < (2p)^{-m/n}$. Let $\chi$ be a distribution over $\mathbb{Z}_q$ such that for all $x$ in the support of $\chi$ we have $x \leq \beta$. Then, the secrets of $\mathsf{LWR}_{n,q,p,\chi}$ are unique with overwhelming probability.*

The construction of the LWR-based LIT is, as previously mentioned, very similar to the LWE-based construction of [9] with errors replaced by rounding.

**Construction 82** *For $q$ prime, $n, p, \beta > 0$ with $p < q$ and $0 < \beta < q$. Choose $m$ such that $4\beta < (2p)^{-m/n}$. Let $\chi$ be a distribution over $\mathbb{Z}_q$ such that for all $x$ in the support of $\chi$ we have $x \leq \beta$. For a uniform $\mathbf{A} \hookleftarrow \$ \, \mathbb{Z}_q^{n \times m}$, let $f_{\mathbf{A}} : \mathbb{Z}_q^n \to \mathbb{Z}_q^m, \mathbf{x} \mapsto \lfloor \mathbf{x}^t \mathbf{A} \rfloor_p$. Then, the LIT $\Pi_{\mathsf{LIT}}$ is defined as in Fig. 24.*

We now need to prove that our construction has the four required security properties tag-indistinguishability (Def. 14), non-invertability (Def. 17), linkability (Def. 15), and unforgeability (Def. 16). Since the proofs are very similar to the proofs for the module LWE-based LIT of [9], we only give an outline for each one.

$\underline{\mathsf{KGen}(1^\lambda)}$

$\mathbf{s} \leftarrow\!\!\$\ \chi^k$

$sk \leftarrow \mathbf{s}$

$\underline{\mathsf{Tag}(sk, \mu)}$

$\mathbf{A}_\mu \leftarrow\!\!\$\ \mathcal{RO}(\mu) \in \mathbb{Z}_q^{k \times m}$

$\mathbf{b}^t \leftarrow \lfloor \mathbf{s}^t \mathbf{A} \rceil_p$

$t \leftarrow \mathbf{b}$

**return** $t$

$\underline{\mathsf{Vrfy}(sk, \mu, t)}$

$\mathbf{A}_\mu \leftarrow\!\!\$\ \mathcal{RO}(\mu) \in \mathbb{Z}_q^{k \times m}$

$\hat{\mathbf{b}}^t \leftarrow \lfloor \mathbf{s}^t \mathbf{A} \rceil_p$

**if** $t = \hat{\mathbf{b}}$

  **return** 1

**return** 0

$\underline{\mathsf{Link}(\mu, t_0, t_1)}$

**if** $t_0 = t_1$

  **return** 1

**return** 0

Fig. 24: Learning with Errors-based LIT.

**Lemma 83.** *Construction 82 has tag-indistinguishability in the random oracle model if* $\mathsf{LWR}_{n,q,p,\chi}$ *is hard.*

To prove this, one uses the LWR assumption twice to replace the public keys and tags generated with $sk_0$ and $sk_1$ by uniformly random values, once in $\mathsf{Anon}^{\mathsf{LIT}}_{\Pi_{\mathsf{LIT}},\mathcal{A},0}(\lambda)$ and once in $\mathsf{Anon}^{\mathsf{LIT}}_{\Pi_{\mathsf{LIT}},\mathcal{A},1}(\lambda)$. Then, the modified games are equal and thus the adversary has advantage 0 to distinguish between them.

**Lemma 84.** *Construction 82 has non-invertability if* $\mathsf{sLWR}_{n,q,p,\chi}$ *is hard.*

To prove this, one uses LWR samples to generate the public key and the tag oracle answers. When the adversary returns an $sk'$, we know by Corollary 81 that the LWR secrets are unique, meaning we have $sk' = sk$, therefore $sk'$ is a solution for the LWR challenge.

**Lemma 85.** *Construction 82 is linkable in the random oracle model.*

In order for the adversary to be able to win, he needs to output $sk_0, sk_1$ such that $f(sk_0) = f(sk_1)$. By Corollary 81 we know that LWR secrets are unique, thus we have $sk_0 = sk_1$. Now the tags $t_0, t_1$ output by the adversary must both verify under $pk_0$ for the message $\mu$ also output by it. Since tagging is deterministic, this can only be if $t_0 = t_1$, thus we have that $t_0, t_1$ link and the adversary always loses.

**Lemma 86.** *Construction 82 is unforgeable in the random oracle model if* $\mathsf{sLWR}_{n,q,p,\chi}$ *is hard.*

To prove this, one uses LWR samples to generate the public key and the tag oracle answers. When the adversary outputs some $sk^*, \mu, t^*)$, we know $t = t^*$ since $\mathsf{Linkable}^{\mathsf{LIT}}(\mu, t, , t^*) = 1$ Furthermore, by Corollary 81 we know that LWR secrets are unique, thus we know that $sk^* = sk_0$ and $sk^*$ is a solution to the LWR challenge.

### F.2 Proof of Theorem 30

*Proof.* Let $\mathcal{A}$ be an adversary against the full collision-resistance of the construction. Let $s' = \omega(\sqrt{\log m})$. We define an alternative security game for full collision-resistance called $\mathsf{Game}_1$ in Fig. 25, where we neither need the trapdoor of the commitment scheme nor the trapdoor $\mathsf{td}$ of $\mathbf{A}$. This is achieved by leveraging the power of the random oracle, i.e., by being able to program the random oracle. In this game, we also remember some additional values, which will be used to define the event $\mathsf{Free}$ mentioned above and to extract either an SIS solution or to break the binding of the commitment scheme. In the original security game, we can implement the random oracle by choosing uniform outputs and storing what we answered so far in a set $\mathcal{Q}_{\mathcal{RO}}$. In $\mathsf{Game}_1$, if we answer a random oracle query, we generate $\mathbf{B} = \mathbf{AU}$, i.e., we do not put a trapdoor into $\mathbf{B}$. We indicate that by also storing $\mathbf{U}$ and an indicator $\mathfrak{td}_{no}$. If, on the other hand, we are answering an $\mathsf{Adapt}\mathcal{O}$ oracle call, we generate $\mathbf{B} = \mathbf{AR} + \mathbf{G}$ with a trapdoor. Again, we indicate this, this time by storing $\mathbf{R}$ and an indicator $\mathfrak{td}_{yes}$.

In the original game, $\mathcal{Q}$ only stored for which $(h, \mu)$ the adversary has seen a collision. In $\mathsf{Game}_1$, $\mathcal{Q}$ also remembers if the $(h, \mu)$ is "fresh", denoted by $\mathfrak{f}$: if the adversary wants $\mathsf{Adapt}\mathcal{O}$ to not return $\perp$, it needs to call the oracle with a valid $(h, \mu)$ pair as the first half of the argument. If the adversary has not seen a collision on this $(h, \mu)$, it is marked as fresh. If that is the case, $\mathcal{Q}$ also remembers the $r$ that was used to generate the $h$ from $\mu$. Pairs $(h, \mu')$ for which the adversary has seen a collision before (this includes the $\mu'$ of a $\mathsf{Adapt}\mathcal{O}$ query) are marked as "old", denoted by $\mathfrak{o}$. Note that by this definition, for every old $(h, \mu')$, there must exist a fresh $(h, \mu)$, because in order for the adversary to see the first collision on $h$, it has to query $\mathsf{Adapt}\mathcal{O}$ with a valid hash, which is then considered fresh.

We can argue that the advantages of $\mathcal{A}$ in the original and the alternative game are negligibly close. This is due to Theorem 56, Theorem 9, and due to the DEE property of the commitment scheme, as well as $z$ being chosen from a superpoly large space (such that no $z$ is repeated with overwhelming probability). For a similar proof, see [31].

We now define the event $\mathsf{Free}$, by the following four subevents:

$\mathsf{FirstFree}$ is true if there exist $\mathbf{B}, \mathbf{U}$ s.t. $((\mu, z, c), \mathbf{B}, \mathbf{U}, \mathfrak{td}_{no}, \perp) \in \mathcal{Q}_{\mathcal{RO}}$, where $(\cdot, \mu, r = (z, \mathbf{e}, c, d), \cdot, \cdot)$ is output by $\mathcal{A}$.

$\mathsf{SecondQueried}$ is true if $(h, \mu') \in \mathcal{Q}$, where $(h, \cdot, \cdot, \mu', \cdot)$ is output by $\mathcal{A}$.

$\mathsf{SecondFree}$ is true if there exist $\mathbf{B}', \mathbf{U}'$ s.t. $((\mu', z', c'), \mathbf{B}', \mathbf{U}', \mathfrak{td}_{no}, \perp) \in \mathcal{Q}_{\mathcal{RO}}$, where $(\cdot, \cdot, \cdot, \mu', r' = (z', \mathbf{e}', c', d'))$ is output by $\mathcal{A}$.

$\mathsf{FreshFree}$ is true if there exist $\hat{\mu}, \hat{z}, \hat{c}, \hat{\mathbf{B}}, \hat{\mathbf{U}}, \hat{\mathbf{e}}$ s.t. $((\hat{\mu}, \hat{z}, \hat{c}), \hat{\mathbf{B}}, \hat{\mathbf{U}}, \mathfrak{td}_{no}, \perp) \in \mathcal{Q}_{\mathcal{RO}}$ and $(h, \hat{\mu}, \mathfrak{f}, \hat{r} = (\hat{z}, \hat{\mathbf{e}}, \hat{c}, \hat{d})) \in \mathcal{Q}$, where $(h, \cdot, \cdot, \cdot, \cdot)$ is output by $\mathcal{A}$.

Then, we can define

$$\mathsf{Free} = \mathsf{FirstFree} \wedge (\neg\mathsf{SecondQueried} \wedge \mathsf{SecondFree} \vee \mathsf{SecondQueried} \wedge \mathsf{FreshFree}).$$

We will argue that if $\mathsf{Free}$ holds and the adversary wins, we can reduce to SIS. The idea is if $\mathsf{Free}$ holds, we can find a collision $((\mu, r), (\mu', r'))$ such that $\mathbf{B}, \mathbf{B}'$,

$\mathsf{Game}_1(1^\lambda)$

$\mathcal{Q}, \mathcal{Q}_{\mathcal{RO}} \leftarrow \emptyset$
$\mathbf{A} \leftarrow\!\$\ \mathbb{Z}_q^{n\times m}$
$(\mathsf{pp}, \cdot) \leftarrow\!\$\ \mathsf{TdGen}(1^\lambda)$
$\mathsf{pk}_{\mathsf{ch}} \leftarrow (\mathbf{A}, \mathsf{pp})$
$(h, \mu, r, \mu', r') \leftarrow\!\$\ \mathcal{A}^{\mathcal{RO}', \mathsf{Adapt}\mathcal{O}'}(\mathsf{pk}_{\mathsf{ch}})$
**return** $(\mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, h, \mu, r) = 1$
$\quad \wedge\ \mathsf{CHashCheck}(\mathsf{pk}_{\mathsf{ch}}, h, \mu', r') = 1$
$\quad \wedge\ \mu \neq \mu'$
$\quad \wedge\ (h, \mu, \cdot, \cdot) \notin \mathcal{Q})$

$\mathcal{RO}'(\cdot)$ with $x \in \{0,1\}^*$

**if** $\exists (x, \mathbf{B}, \cdot, \cdot) \in \mathcal{Q}_{\mathcal{RO}}$
**return** $\mathbf{B}$
$\mathbf{U} \leftarrow\!\$\ D_{\mathbb{Z}^{2m}, s'}$
$\mathbf{B} \leftarrow\!\$\ \mathbf{AU}$
$\mathcal{Q}_{\mathcal{RO}} \leftarrow \mathcal{Q}_{\mathcal{RO}} \cup \{(x, \mathbf{B}, \mathbf{U}, \mathfrak{td}_{no}, \bot)\}$
**return** $\mathbf{B}$

$\mathsf{Adapt}\mathcal{O}'(\mathsf{sk}_{\mathsf{ch}}, \cdot, \cdot, \cdot, \cdot)$ with $(h, \mu, r, \mu')$

**if** $\mathsf{CHashCheck}(pk, h, \mu, r) = 0$
**return** $\bot$
$z' \leftarrow\!\$\ \{0,1\}^\lambda$
$\mathbf{R} \leftarrow\!\$\ D_{\mathbb{Z}^{2m}, s'}$
$\mathbf{B} \leftarrow\!\$\ \mathbf{AR} + \mathbf{G}$
$\mathbf{e}' \leftarrow\!\$\ \mathsf{PreSample}(\left[\mathbf{A}\ \mathbf{B}\right], \mathbf{R}, h, s)$
$(c', d') \leftarrow\!\$\ \mathsf{Com}(\mathsf{pp}, \mathbf{e}')$
$x \leftarrow (\mu', z', c')$
$r' \leftarrow (z', \mathbf{e}', c', d')$
$\mathcal{Q}_{\mathcal{RO}} \leftarrow \mathcal{Q}_{\mathcal{RO}} \cup \{(x, \mathbf{B}, \mathbf{R}, \mathfrak{td}_{yes}, r')\}$
**if** $\nexists (h, \mu, \cdot, \cdot) \in \mathcal{Q}$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(h, \mu, \mathfrak{f}, r)\}$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(h, \mu', \mathfrak{o}, \bot)\}$
**return** $r'$

Fig. 25: The alternative security game $\mathsf{Game}_1$.

which are defined by the first and second pair of the collision, respectively, do not contain a gadget $\mathbf{G}$. This collision always includes the $(\mu, r)$ output by the adversary, but the second element is either the $(\mu', r')$ output by the adversary or the fresh pair $(\hat{\mu}, \hat{r})$ if $(\mu', r')$ is old. Then, we can combine the collision into a solution for SIS. On the other hand, if $\mathsf{Free}$ does not hold, we are able to break the binding of the commitment scheme. To show this, we construct two adversaries, $\mathcal{B}$ against SIS in Fig. 26 and $\mathcal{C}$ against the binding of the commitment scheme in Fig. 27 that both simulate $\mathcal{A}$.

Let $(h, \mu, r, \mu', r')$ be the forgery output by $\mathcal{A}$ with $r = (z, \mathbf{e}, c, d)$ and $r' = (z', \mathbf{e}', c', d')$. If $\mathsf{Free}$ happens, we can easily see that $\mathcal{B}$ is able to find $\mathbf{B}, \mathbf{e}, \hat{\mathbf{B}}, \hat{\mathbf{e}}$, where the matrices are free of $\mathbf{G}$ and such that

$$\left[\mathbf{A}\ \mathbf{B}\right] \mathbf{e} = \left[\mathbf{A}\ \hat{\mathbf{B}}\right] \hat{\mathbf{e}}.$$

Therefore, we can easily see that $\mathbf{Av} = \mathbf{0}$ and that

$$\begin{aligned} \|\mathbf{v}\| =& \left\| \left[\mathbf{I}\ \mathbf{U}\right] \mathbf{e} - \left[\mathbf{I}\ \hat{\mathbf{U}}\right] \hat{\mathbf{e}} \right\| \\ \leq& (s_1(\mathbf{U}) + 1) \cdot \beta + (s_1(\hat{\mathbf{U}}) + 1) \cdot \beta \\ \leq& 2s' \cdot (\sqrt{n} + \sqrt{m} + t + 1) \cdot s\sqrt{2m} \end{aligned}$$

$\mathcal{B}(\mathbf{A})$

$\mathcal{Q}, \mathcal{Q}_{\mathcal{RO}} \leftarrow \emptyset$

$(\mathsf{pp}, \cdot) \leftarrow\!\!\$\ \mathsf{TdGen}(1^\lambda)$

$\mathsf{pk}_{\mathsf{ch}} \leftarrow (\mathbf{A}, \mathsf{pp})$

$(h, \mu, r, \mu', r') \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{RO}', \mathsf{Adapt}\mathcal{O}'}(\mathsf{pk}_{\mathsf{ch}})$

with $r = (z, \mathbf{e}, c, d), r' = (z', \mathbf{e}', c', d')$

**if** FirstFree

Let $\mathbf{B}, \mathbf{U}$ be s.t. $((\mu, z, c), \mathbf{B}, \mathbf{U}, \mathfrak{td}_{no}, \bot) \in \mathcal{Q}_{\mathcal{RO}}$

**if** SecondQueried $\wedge$ SecondFree

Let $\mathbf{B}', \mathbf{U}'$ be s.t. $((\mu', z', c'), \mathbf{B}', \mathbf{U}', \mathfrak{td}_{no}, \bot) \in \mathcal{Q}_{\mathcal{RO}}$

**return** $\mathbf{v} \leftarrow \left[\mathbf{I}\ \mathbf{U}\right] \mathbf{e} - \left[\mathbf{I}\ \mathbf{U}'\right] \mathbf{e}'$

**if** $\neg$SecondQueried $\wedge$ FreshFree

Let $\hat{\mu}, \hat{z}, \hat{c}, \hat{\mathbf{B}}, \hat{\mathbf{U}}, \hat{\mathbf{e}}$ be s.t. $(h, \hat{\mu}, \mathfrak{f}, (\hat{z}, \hat{\mathbf{e}}, \hat{c}, \cdot)) \in \mathcal{Q}$

and $((\hat{\mu}, \hat{z}, \hat{c}), \hat{\mathbf{B}}, \hat{\mathbf{U}}, \mathfrak{td}_{no}, \bot) \in \mathcal{Q}_{\mathcal{RO}}$

**return** $\mathbf{v} \leftarrow \left[\mathbf{I}\ \mathbf{U}\right] \mathbf{e} - \left[\mathbf{I}\ \hat{\mathbf{U}}\right] \hat{\mathbf{e}}$

**return** $\bot$

Fig. 26: Adversary against SIS.

by definition of $\beta$ and Theorem 57. Furthermore, by definition of CHashCheck we know that $\mathbf{0} \neq \mathbf{e}_2, \hat{\mathbf{e}}_2 \in \mathbb{Z}_q^m$, where $(\mathbf{e}_1^t, \mathbf{e}_2^t)^t = \mathbf{e}^t$, $(\hat{\mathbf{e}}_1^t, \hat{\mathbf{e}}_2^t)^t = \hat{\mathbf{e}}^t$. Due to this fact, together with the distributions of $\mathbf{U}, \hat{\mathbf{U}}$ having high min-entropy and a standard entropy argument (see [31] for an example), we can argue that $\mathbf{v} \neq \mathbf{0}$. Thus, $\mathbf{v}$ is an SIS solution and we know that

$$\Pr[\mathsf{f\text{-}CR}_{\mathsf{CH},\mathcal{A}}(1^\lambda) = 1 \wedge \mathsf{Free}] = \mathsf{SIS}_{n,m,q,\beta'}(\mathcal{B}).$$

If $\neg$Free happens, we make a case distinction which part of the event is not true.

**Case 1 ($\neg$FirstFree):** In this case, we know that $\nexists((\mu, z, c), \mathbf{B}, \mathbf{R}, \mathfrak{td}_{no}, \bot) \in \mathcal{Q}_{\mathcal{RO}}$. We assume wlog. that the adversary queries the random oracle (or by extension, Adapt$\mathcal{O}$) to get any $\mathbf{B}$, because else it has to predict the random oracle. Since $\nexists((\mu, z, c), \mathbf{B}, \mathbf{R}, \mathfrak{td}_{no}, \bot) \in \mathcal{Q}_{\mathcal{RO}}$ but $\mathcal{A}$ has output a valid hash on $\mu$, we know that $\mathbf{B}$ must have been generated in an Adapt$\mathcal{O}$ call. Therefore $\exists((\mu, z, c), \mathbf{B}, \mathbf{R}, \mathfrak{td}_{yes}, r^*) \in \mathcal{Q}_{\mathcal{RO}}$ which $\mathcal{B}$ can find. Furthermore, since that tuple was generated during an Adapt$\mathcal{O}$ call, we know that $(\mathbf{B}, \mathbf{e}^*, \mu, \cdot, \cdot) \in \mathcal{Q}$. However, by definition of the security game we know that $(h, \mu, \cdot, \cdot) \notin \mathcal{Q}$. Therefore $\mathbf{e} \neq \mathbf{e}^*$ and $\mathcal{C}$ has broken the binding of the commitment.

**Case 2 ($\neg$SecondQueried $\wedge$ $\neg$SecondFree):** In this case, since $(h, \mu', \cdot, \cdot) \notin \mathcal{Q}$ we argue as in Case 1.

**Case 3 (SecondQueried $\wedge$ $\neg$FreshFree):** In the case that $(h, \mu', \cdot, \cdot) \in \mathcal{Q}$ we first argue that there must exist some $(h, \hat{\mu}, \mathfrak{f}, \hat{r}) \in \mathcal{Q}$. Either $(h, \mu', \mathfrak{f}, \hat{r}) \in \mathcal{Q}$ and

$\mathcal{C}(\mathsf{pp})$

---

$\mathcal{Q}, \mathcal{Q}_{\mathcal{RO}} \leftarrow \emptyset$

$\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n \times m}$

$\mathsf{pk}_{\mathsf{ch}} \leftarrow (\mathbf{A}, \mathsf{pp})$

$(h, \mu, r, \mu', r') \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{RO}', \mathsf{Adapt}\mathcal{O}'}(\mathsf{pk}_{\mathsf{ch}})$

  with $r = (z, \mathbf{e}, c, d), r' = (z', \mathbf{e}', c', d')$

**if** $\neg\mathsf{FirstFree}$

  Let $\mathbf{e}^*, d^*$ be s.t. $((\mu, z, c), \cdot, \cdot, \mathfrak{td}_{yes}, (z, \mathbf{e}^*, c, d^*)) \in \mathcal{Q}_{\mathcal{RO}}$

  **return** $(c, \mathbf{e}, d, \mathbf{e}^*, d^*)$

**if** $\mathsf{SecondQueried} \wedge \neg\mathsf{SecondFree}$

  Let $\mathbf{e}^*, d^*$ be s.t. $((\mu', z', c'), \cdot, \cdot, \mathfrak{td}_{yes}, z', \mathbf{e}^*, c', d^*) \in \mathcal{Q}_{\mathcal{RO}}$

  **return** $(c', \mathbf{e}', d', \mathbf{e}^*, d^*)$

**if** $\neg\mathsf{SecondQueried} \wedge \neg\mathsf{FreshFree}$

  Let $\hat{\mu}, \hat{z}, \hat{c}, \hat{\mathbf{e}}, \hat{d}, \bar{h}, \bar{\mathbf{e}}, \bar{d}$ be s.t. $(h, \hat{\mu}, \mathfrak{f}, \cdot) \in \mathcal{Q}$

    such that $((\hat{\mu}, \hat{z}, \hat{c}), \cdot, \cdot, \mathfrak{td}_{yes}, (\hat{z}, \hat{\mathbf{e}}, \hat{c}, \hat{d})) \in \mathcal{Q}_{\mathcal{RO}}$

    and $(\bar{h}, \hat{\mu}, \mathfrak{f}, (\hat{z}, \bar{\mathbf{e}}, \hat{c}, \bar{d})) \in \mathcal{Q}$

  **return** $(\hat{c}, \hat{\mathbf{e}}, \hat{d}, \bar{\mathbf{e}}, \bar{d})$

**return** $\perp$

Fig. 27: Adversary against the binding of the commmitment scheme.

we are done. Else we have added $(h, \mu', \mathfrak{o}, \perp)$ to $\mathcal{Q}$ since $(h, \mu', \cdot, \cdot) \in \mathcal{Q}$, which happened during an $\mathsf{Adapt}\mathcal{O}(h, \mu^*, r^*, \mu')$ query. This means that $\mu'$ was the message for which $\mathsf{Adapt}\mathcal{O}$ was supposed to find a collision, i.e., it was the second message during the oracle call. During that same oracle call either $(h, \mu^*, \mathfrak{f}, r^*)$ was added to $\mathcal{Q}$, or it was already the case that $(h, \mu^*, \cdot, \cdot) \in \mathcal{Q}$. In the latter case, we can recursively apply the above argument until we find $(h, \hat{\mu}, \mathfrak{f}, \hat{r})$.

Once we have $(h, \hat{\mu}, \mathfrak{f}, \hat{r})$, since $\neg\mathsf{FreshFree}$ holds and by our assumption that $\mathcal{A}$ does not try to predict the random oracle, we know that $\mathcal{RO}(\hat{\mu}, \hat{z}, \hat{c}) = \hat{\mathbf{B}} = \mathbf{A}\hat{\mathbf{R}} + \mathbf{G}$. But since $\hat{\mathbf{B}}$ has this form, it must have been created during an $\mathsf{Adapt}\mathcal{O}$ call. Because we argued that $(h, \hat{\mu}, \mathfrak{f}, \cdot)$ is fresh, this must mean when $\hat{\mathbf{B}}$ was created, it was created for some different hash $\bar{h}$. Thus, there exists some $((\bar{\mu}, \bar{z}, \bar{c}), \hat{\mathbf{B}}, \cdot, \mathfrak{td}_{yes}, \bar{r}) \in \mathcal{Q}_{\mathcal{RO}}$ with $\hat{r} = (\hat{z}, \hat{e}, \hat{c}, \hat{d}) \neq \bar{r} = (\hat{z}, \bar{e}, \bar{c}, \bar{d})$. However, due to $\mathbf{A}\hat{\mathbf{R}}$ being statistically close to uniform, we have with overwhelming probability that the inputs to the oracle that correspond to $\hat{h}, \bar{h}$ were equal, i.e., $\hat{\mu} = \bar{\mu}, \hat{z} = \bar{z}, \hat{c} = \bar{c}$. Now since $h = \hat{\mathbf{B}}\hat{\mathbf{e}} \neq \hat{\mathbf{B}}\bar{\mathbf{e}} = \bar{h}$, we know that $\hat{\mathbf{e}} \neq \bar{\mathbf{e}}$, which are the messages committed in $\hat{c}, \bar{c}$ respectively. Thus, $\mathcal{C}$ breaks the binding of the commitment scheme. Therefore, we know that

$$\Pr[\mathsf{f\text{-}CR}_{\mathsf{CH},\mathcal{A}}(1^\lambda) = 1 \wedge \neg\mathsf{Free}] = \Pr[\mathsf{Bind}_{\Pi_{\mathsf{Com}},\mathcal{C}}(\lambda) = 1]$$

and thus

$$\Pr[\mathsf{f\text{-}CR}_{\mathsf{CH},\mathcal{A}}(1^\lambda)=1] = \mathsf{SIS}_{n,m,q,\beta'}(\mathcal{B}) + \Pr[\mathsf{Bind}_{\Pi_{\mathsf{Com}},\mathcal{C}}(\lambda)=1]. \qquad \square$$

### F.3 Proof of Theorem 46

*Proof.* For simplicity, we only consider the case of two blocks. The adversary against signer accountability is given a public key $pk_{\mathsf{San}}$ of a sanitizer and has access to a sanitize oracle.

The attacker honestly generates a key pair $(pk_{\mathsf{Sig}}, sk_{\mathsf{Sig}})$ of the signature scheme $\Sigma$. The attacker picks randomly a message $\mu = (\mu_1, \mu_2)$, another message $\mu'[2]$ such that $\mu_2 \neq \mu'[2]$ and sets $\mu' = (\mu_1, \mu'[2])$, a random NONCE, and an auxiliary value $r[2]$. Then, TAG is set to $\mathsf{PRG}(x)$ where $x \coloneqq \mathsf{PRF}(\kappa, \text{NONCE})$, ADM is defined to contain the information that only block 2 is admissible for change, MOD is set such that $\text{MOD}(\mu) = \mu'$. Finally, it sets $\sigma_0$, the underlying signature, to $\mathsf{Sign}(sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \text{ADM}, \text{TAG}, \mu, r[2])$.

The attacker queries the sanitize oracle with message $\mu$, modifications MOD, and signature $(\sigma_0, \text{TAG}, \text{NONCE}, \text{ADM}, r[2])$, and receives NONCE′, TAG′, and $r'[2]$, which constitute the new signature. In particular, it holds that

$$\begin{aligned}&\mathsf{CH}(pk_{\mathsf{San}}, \text{TAG}, (2, \mu_2, pk_{\mathsf{Sig}}), r[2]) = \\ &\mathsf{CH}(pk_{\mathsf{San}}, \text{TAG}', (2, \mu'[2], pk_{\mathsf{Sig}}), r'[2]). \end{aligned} \tag{1}$$

The attacker returns $pk^*_{\mathsf{Sig}} = pk_{\mathsf{Sig}}$, $\mu^* = (\mu'[1], \mu'[2])$, for some $\mu'[1] \neq \mu_1$, $\text{TAG}^* = \text{TAG}'$, $r^*[2] = r'[2]$, $\text{NONCE}^* = \text{NONCE}'$, $\sigma^*$ the signature $\mathsf{Sign}(sk_{\mathsf{Sig}}, pk_{\mathsf{San}}, \text{ADM}, \text{TAG}^*, \mu^*, r^*[2])$, and the proof $\pi^*$ given by $pk_{\mathsf{Sig},\pi} = pk_{\mathsf{Sig}}$, $\text{TAG}_\pi = \text{TAG}$, $(2, \mu_\pi[2], pk_{\mathsf{Sig}})$, where $\mu_\pi[2] = \mu_2$, $r_\pi[2] = r[2]$ and $x_\pi = x$.

The attacker wins, as the following conditions are satisfied:

(i) $\mu^* \neq \mu$ and $\mu^* \neq \mu'$,
(ii) $\mathsf{Vrfy}(pk^*_{\mathsf{Sig}}, pk_{\mathsf{San}}, \mu^*, \sigma^*) = 1$ as the $\mu^*$ was honestly signed,
(iii) $\mathsf{Judge}(\mu^*, \sigma^*, pk^*_{\mathsf{Sig}}, pk_{\mathsf{San}}, \pi^*) = \mathsf{Sanit}$,

where the final step holds as

(a) $pk^*_{\mathsf{Sig}} = pk_{\mathsf{Sig}} = pk_{\mathsf{Sig},\pi}$,
(b) $\mathsf{CH}(pk_{\mathsf{San}}, \text{TAG}^*, (2, \mu^*[2], pk^*_{\mathsf{Sig}}), r^*[2])$
$= \mathsf{CH}(pk_{\mathsf{San}}, \text{TAG}_\pi, (2, \mu_\pi, pk_{\mathsf{Sig},\pi}), r_\pi[2])$,
(c) Block 2 is admissible,
(d) $\text{TAG}_\pi = \mathsf{PRG}(x_\pi)$.

Indeed, (a), (c), and (d) hold by definition, (b) is Equation (1). $\square$

*Remark 87.* The attacker does not need to construct a new hash collision, but makes use of a collision produced by the sanitizer. The reason the present construction is vulnerable is that it suffices to modify the message in a single block to ensure condition (i), although the judge only tests a single block for collisions. Thus, the attacker can modify the message in one block to pass the check in condition (i), and use a collision generated by the sanitizer to convince the judge in condition (iii).