Sanitizable Signatures with Different Admissibility Policies for Multiple Sanitizers

Full Version

Osama Allabwani[1,3][0009-0000-1412-5857], Olivier Blazy[2][0000-0001-6205-8249], Pascal Lafourcade[1,4][0000-0002-4459-511X], Charles Olivier-Anclin[1][0000-0002-9365-3259], and Olivier Raynaud[1][0000-0002-9768-5477]

[1] Université Clermont Auvergne, LIMOS, CNRS, Clermont-Ferrand, France
[2] École Polytechnique, Palaiseau, France
[3] BeYs, Clermont-Ferrand, France
[4] ASTEROIDE, Trust4Sign, La Monnerie-le-Montel, France

Abstract. Sanitizable signatures authorize semi-trusted sanitizers to modify admissible blocks of a signed message. Most works consider only one sanitizer while those considering multiple sanitizers are limited by their capacity to manage admissible blocks which must be the same for all of them. We study the case where different sanitizers with different roles can be trusted to modify different blocks of the message. We define a model for multi-sanitizer sanitizable signatures which allow managing authorization for each sanitizer independently. We also provide formal definitions of its security properties. We propose two secure generic constructions FSV-k-SAN and IUT-k-SAN with different security properties. We implement both constructions and evaluate their performance on a server and a smartphone.

1 Introduction

Sanitizable signatures introduced by Ateniese *et al.* [2] allow an authorized semi-trusted sanitizer to modify parts of a signed message and its corresponding signature while keeping it valid without interacting with the original signer. An *admissibility policy* specifies exactly which parts of the message are admissible for modification. Sanitizable signatures have multiple security properties: *immutability* which requires that the sanitizer cannot modify an inadmissible message block [2], *accountability* which ensures that the signer or sanitizer cannot accuse the other party of signing a certain message [2], *privacy* which requires that the original message does not leak from the sanitized signature [2], *transparency* which guarantees indistinguishability between sanitized and fresh signatures [2], *invisibility* which aims to keep the class of admissible blocks hidden from external parties [2], and *unlinkability* which prevents linking sanitized signatures to their sources [8]. Most existing works consider only having one sanitizer while

having multiple sanitizers can be necessary for certain use cases. This was studied in several works [1, 7, 14, 15, 25, 30]. In this case a new security property, called *sanitizer anonymity*, is considered which maintains the anonymity of the sanitizer who did the sanitization within the list of authorized sanitizers [14]. However, in all of these works, all sanitizers have the same admissibility policy. Our aim is to have a different policy for each sanitizer.

We describe two of several use cases that benefit from having multiple sanitizers where different sanitizers are authorized to modify different blocks. Suppose that we want to sign a multi-party contract such as a house rental contract that should be filled and signed by a real estate agent, a landlord, a tenant, and a guarantor. The real estate agent will be the signer while the other parties are sanitizers. The real estate agent prepares the contract by filling the information of the house, the landlord, and the rental price and signs it. Then, the tenant and guarantor fill their information and add their signature to the contract. Finally, the landlord fills any additional information related to him and adds his signature. As each party needs to fill a specific section of the contract, we need different admissible blocks for each of them. In traditional sanitizable signatures, the signature is valid directly after being generated by the signer which gives him the ability to ignore certain sanitizers and fill their information himself. Thus, we introduce a new property called *full-sanitization verifiability* which requires that a signature is valid if and only if all admissible blocks were later verified/modified by the sanitizers.

As a second use case, consider the scenario of a digital identity card that contains the name, birthdate, address, and the highest level of education obtained. The card is generated by a digital identity provider, but the information is maintained by trusted third parties: the government is responsible for the name and birthdate, electricity and internet providers for the address, and educational institutions for the education level. This means that we require different admissible blocks for each of them. In this setting, we need also full-sanitization verifiability as the digital identity provider is not trusted to provide any modifiable blocks. Additionally, the sanitizer anonymity property is required to hide which electricity or internet provider the person has a contract with, or which university or school they attended.

Our Contributions. We introduce a new primitive called *k-sanitizer sanitizable signature* (k-SAN) which authorizes different sanitizers to modify different blocks. We formalize k-SAN and its security where we have one signer and k sanitizers and each sanitizer can only modify a subset of the admissible blocks. Our security model retains traditional security properties of sanitizable signatures, but we consider the proof-restricted variant of transparency from [24] instead of transparency as defined in [2]. We also introduce a new security property, *full-sanitization verifiability* which requires that a signature is valid if and only if all admissible blocks were sanitized, *i.e.,* modified/verified by a sanitizer. Having full-sanitization verifiability breaks proof-restricted transparency and invisibility which we prove in Section 3. Thus, we design two generic constructions for

k-SAN with different security properties required by different use cases which is our second contribution.

The first construction is called Full-Sanitization-Verifiable k-SAN (FSV-k-SAN). It uses a chameleon hash [23] and has the following security properties: immutability, accountability, privacy, sanitizer anonymity, and full-sanitization verifiability. This construction can be used for the digital identity card and multi-party contract use cases described above.

The second construction is called Invisible-Unlinkable-Transparent k-SAN (IUT-k-SAN). It leverages ideas from the work of Bultel *et al.* [11] by using two types of re-randomizable signatures [11, 22]. It has the same security properties as the first construction, but we replace full-sanitization verifiability with proof-restricted transparency, unlinkability, and invisibility. IUT-k-SAN can be used for the following use case. A patient does a medical test in a hospital and the doctor signs the test result. The document containing the test results can be used by different departments in the hospital for different purposes. For example, the accounting department needs to send the document to the patient's insurance provider for billing, while the research department needs to send the document to a third party for data analysis. The signer authorizes the accounting department to sanitize the document in order to hide the test results before sending it to the insurance provider, and allows the research department to hide the accounting and personal details before sending it to the data analysis third party. As discussed in [8, 11], unlinkability is useful here to prevent someone who has access to both the insurance provider's database and the data analysis database from linking the stored files to the same signature and reconstructing the original document. The authors of [11] also argue that invisibility can be useful if the patient is given the right to modify some test results. For example, a patient who tested positive for a disease is given the possibility to change the result to negative to avoid discrimination, while a patient who tested negative is not allowed to change the result to positive to prevent him from asking for the associated medication. Invisibility and proof-restricted transparency are important to ensure that the receiver of the document cannot predict the real test result by knowing if the patient is allowed to modify it or if the document was sanitized.

In Table 1, we show the security properties supported by each construction and compare them to existing multi-sanitizer schemes [1, 7, 14, 15, 25, 30] and in particular Canard *et al.*'s work [14] as it supports the most properties. IUT-k-SAN improves the state of the art of the multi-sanitizer setting by satisfying invisibility in addition to all the properties from [14]. The construction does not satisfy full-sanitization verifiability because it breaks proof-restricted transparency and invisibility as explained before.

Our generic constructions show how some existing sanitizable signature schemes can be transformed to support k sanitizers, but this does not work on all existing schemes without losing security properties. For example, in [12], the authors

Table 1: Comparison of the security properties of our k-SAN constructions and other multi-sanitizer schemes.

Security Property	FSV-k-SAN	IUT-k-SAN	[1]	[7]	[14]	[15]	[25]	[30]
Unforgeability	✓	✓	✓	✓	✓	✓	✓	✓
Immutability	✓	✓	✓	✓	✓	✓	✓	✓
Privacy	✓	✓	✓	✓	✓	✓	✓	✓
Accountability	✓	✓	✓	✓	✓	✗	✓	✓
Sanitizer Anonymity	✓	✓	✗	✗	✓	✗	✗	✓
Unlinkability	✗	✓	✓	✗	✓	✗	✗	✗
Full-Sanitization Verifiability	✓	✗	✗	✗	✗	✗	✗	✗
Transparency	✗	✓	✓	✗	✓	✓	✓	✓
Invisibility	✗	✓	✗	✗	✗	✗	✗	✗

use a Signature of Knowledge (SoK) which needs to be re-randomized to have unlinkability. If we try to add k sanitizers, we need to duplicate the SoK for each sanitizer according to his admissible blocks. However, re-randomization is only possible if we have access to the secret key that allows the sanitizer to do the admissible modifications. Thus, if we share the keys of all SoKs with all sanitizers to do the re-randomization, immutability is lost.

Finally, we implemented both constructions in Rust which is, to the best of our knowledge, the first implementation of multi-sanitizer sanitizable signatures. We also evaluated the implementation's performance on a server and a smartphone. For realistic security parameters, 5 sanitizers, and a message comprised of 15 blocks from which 5 are admissible, signing in both constructions took less than 800 ms on the server and less than 1300 ms on the smartphone. In FSV-k-SAN, verifying took less than 520 ms and 1100 ms on the server and the smartphone respectively. Other algorithms in both constructions took less than 205 ms on the server and less than 410 ms on the smartphone. These performances are acceptable for many real-world applications that require the advanced features and security properties that we propose.

Related works. Sanitizable signatures were introduced by Ateniese *et al.* in [2], who identified several security properties (unforgeability, immutability, privacy, transparency, and accountability) later formally defined in [6]. Invisibility was also introduced in [2] but received formal treatment much later in [13]. Unlinkability has been introduced and formalized in [8] and studied in [9, 21]. Only three schemes guarantee all these properties at once [5, 11, 12]. Two kinds of sanitizer limitation have also been considered in the literature encompassing limits in the number of sanitizations [12] or blocks sanitized in a single sanitization [5]. Sanitizable signatures with several sanitizers have been considered in multiple works [1, 7, 14, 15, 25, 30] but with only one admissibility policy common to

all sanitizers. This setting required also defining new properties such as sanitizer anonymity [14]. In this article, we extend these results and allow the management of different admissibility policies for different sanitizers. This requires rethinking the security model and proofs, for example, immutability traditionally requires protecting against the modification of inadmissible blocks overall, but in our case we have also to protect against the modification of blocks which are admissible only for sanitizers to which the adversary does not have access. In addition, the oracles that do sanitization require the adversary to choose which of the k sanitizers he wants to use. Moreover, adding k sanitizers with different admissible blocks make the constructions more complex and potentially less efficient.

Notation. We use classical cryptographic notation. For a positive integer k, $[\![k]\!] = \{1, \ldots, k\}$. For a set S, $r \leftarrow\!\!\$\, S$ means that r is chosen uniformly at random from S and $|S|$ is its cardinal. We denote by $y \leftarrow \mathsf{Alg}(x)$ the execution of an algorithm Alg outputting y on input x. Considering a second algorithm $\mathcal{O}$ and an algorithm $\mathcal{A}$, $\mathcal{A}^{\mathcal{O}}$ means that the algorithm $\mathcal{A}$ has access to $\mathcal{O}$ as a black-box oracle. $\mathsf{Adv}^{\mathsf{PR}}_{\mathsf{SC},\mathcal{A}}(\lambda)$ denotes the advantage of the adversary $\mathcal{A}$ in breaking the property PR of the scheme SC under the security parameter λ. We denote by $\mathsf{negl}(\lambda)$ a negligible function in the security parameter λ. For a vector of elements $v = (a, b, c)$ we use $v.a$ to refer to the element called a in v. Moreover, $(a, b) \xleftarrow{\mathsf{p}} v$ denotes parsing the tuple v as the two elements a and b. The function $v \leftarrow \mathsf{Append}(v, x)$ appends the element x to the vector v, while $\mathbf{M} \leftarrow \mathsf{Append}(\mathbf{M}, v)$ appends the vector v to the matrix $\mathbf{M}$ as a new row. The function $\mathsf{AppendC}$ does the same but for columns. The notation $\mathbf{M}_i$ refers to the vector representing the i'th row of matrix $\mathbf{M}$ while $\mathbf{M}_{i,j}$ refers to the element in row i and column j.

2 *k*-Sanitizer Sanitizable Signatures

In k-sanitizer sanitizable signatures, a signer S can sign a message m using the Sign algorithm, which outputs a signature σ. The message m is split into n blocks as $m_1 \| \cdots \| m_n$, where each m_i belongs to the message space $\mathcal{M}$. The Sign algorithm uses k sanitizers public keys instead of a single key in traditional sanitizable signatures. Each of the k sanitizers may subsequently modify the signed message and its signature in accordance with his independently defined admissibility policy using the $\mathsf{Sanitize}$ algorithm. Before presenting the formal definition, we introduce the formalism that supports such independent policies in k-sanitizer sanitizable signatures.

We denote the list of the k sanitizers' public keys by the vector $\mathbf{PKZ} = (\mathsf{pk}^i_{\mathsf{Z}})_{i \in [\![k]\!]}$. Each sanitizer with public key $\mathsf{pk}^i_{\mathsf{Z}}$ is either authorized to modify a given message block m_j ($a_{i,j} = 1$), or not ($a_{i,j} = 0$). This permission structure is captured by the *admissibility matrix* $\mathbf{A} = (a_{i,j})_{i \in [\![k]\!],\, j \in [\![n]\!]} \in \{0,1\}^{k \times n}$. According to the constraints defined by the signer through $\mathbf{A}$, the sanitization operation is defined by a modification vector MOD, which consists of a list of pairs (j, m'_j) indicating that the message block m_j is to be replaced by m'_j. Based on $\mathbf{PKZ}$, $\mathbf{A}$, and MOD, we now define the algorithms that constitute a k-SAN scheme.

Definition 1 (k-SAN). *A* k-Sanitizer Sanitizable Signature scheme k-SAN *consists of the following Probabilistic Polynomial Time (*PPT*) algorithms:*

$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda, n)$: *On input the security parameter* λ *and the message size* n*, the algorithm outputs the public parameter* pp *(implicit input for the other algorithms).*

$(\mathsf{sk_S}, \mathsf{pk_S}) \leftarrow \mathsf{KGen_S}(\mathsf{pp})$: *On input the public parameter* pp*, the algorithm outputs a tuple* $(\mathsf{sk_S}, \mathsf{pk_S})$ *containing* $\mathsf{sk_S}$ *the secret key and* $\mathsf{pk_S}$ *the public key of a signer.*

$(\mathsf{sk_Z}, \mathsf{pk_Z}) \leftarrow \mathsf{KGen_Z}(\mathsf{pp})$: *On input the public parameter* pp*, the algorithm outputs a tuple* $(\mathsf{sk_Z}, \mathsf{pk_Z})$ *containing* $\mathsf{sk_Z}$ *the secret key and* $\mathsf{pk_Z}$ *the public key of a sanitizer.*

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk_S}, \mathbf{PKZ}, m, \mathbf{A})$: *On input a signer secret key* $\mathsf{sk_S}$*, a list of sanitizer public keys* $\mathbf{PKZ}$*, a message* m*, and an admissibility matrix* $\mathbf{A}$*, the algorithm outputs a signature* σ*.*

$\sigma' \leftarrow \mathsf{Sanitize}(\mathsf{sk_Z}, \mathsf{pk_S}, \mathbf{PKZ}, m, \mathsf{MOD}, \sigma)$: *On input a sanitizer secret key* $\mathsf{sk_Z}$*, a signer public key* $\mathsf{pk_S}$*, a list of sanitizer public keys* $\mathbf{PKZ}$*, a message* m*, a modification* MOD*, and a signature* σ*, the algorithm outputs a new signature* σ' *on* $\mathsf{MOD}(m)$*.*

$b \leftarrow \mathsf{Verify}(\mathsf{pk_S}, \mathbf{PKZ}, m, \sigma)$: *On input a signer public key* $\mathsf{pk_S}$*, a list of sanitizer public keys* $\mathbf{PKZ}$*, a message* m*, a signature* σ*, the algorithm outputs a bit* b *indicating if the signature is valid.*

A k-SAN *scheme is expected to satisfy* correctness*: any signature produced by the* Sign *algorithm must be accepted as valid by the* Verify *algorithm. In addition, the scheme should ensure the properties of* Unforgeability*,* Immutability*, and* Privacy*, as described in Section 3. Depending on the use case, additional properties may be desirable, such as* Proof-Restricted Transparency*,* Invisibility*,* Unlinkability*,* Full-Sanitization Verifiability*, and* Sanitizer Anonymity*.*

To additionally provide Accountability*, the signer may invoke the* Prove *algorithm, which generates a proof that can be verified using the* Judge *algorithm.*

$\pi \leftarrow \mathsf{Prove}(\mathsf{sk_S}, \mathbf{PKZ}, m, \sigma, j)$: *On input a signer secret key* $\mathsf{sk_S}$*, a list of sanitizer public keys* $\mathbf{PKZ}$*, a message* m*, a signature* σ*, and an index of a message block* j *(index* j *is only used when we have full-sanitization verifiability), the algorithm outputs a proof* π *for the message block* j*. If* $j = \perp$*, the algorithm outputs a proof for the whole message.*

$d \leftarrow \mathsf{Judge}(\mathsf{pk_S}, \mathbf{PKZ}, m, \sigma, \pi, j)$: *On input a signer public key* $\mathsf{pk_S}$*, a list of sanitizer public keys* $\mathbf{PKZ}$*, a message* m*, a signature* σ*, a proof* π*, and an index of a message block* j*, the algorithm outputs a decision* $d \in \{S, Z\}$ *indicating whether the message block* j *was generated by the signer or the sanitizer. If* $j = \perp$*, the algorithm judges the whole message.*

We present two generic constructions, IUT-k-SAN and FSV-k-SAN, each designed for different use cases based on their distinct properties. Their respective security guarantees are listed in Table 1 while being formalized in Section 3.

3 Security Model

Our security formalism builds upon the foundations laid out in [11]. A k-SAN scheme must satisfy *Unforgeability*, *Immutability* and *Privacy*. Furthermore, our two proposed constructions aim to achieve *Signer Accountability*, *Sanitizer Accountability*, and *Sanitizer Anonymity*. In addition, *Proof-Restricted Transparency*, *Invisibility*, and *Unlinkability* are properties specific to IUT-k-SAN, while *Full-Sanitization Verifiability* is satisfied only by FSV-k-SAN.

Before proceeding, we introduce additional notation used to formalize security in the k-sanitizer setting.

Let $\mathbf{A} \in \{0,1\}^{k \times n}$ be an admissibility matrix. Let:

- $\mathsf{ADM}^{\mathbf{A}}(j) = \bigvee_{i=1}^{k} a_{i,j}$ — some sanitizer can modify block j,
- $\mathsf{ADM}^{\mathbf{A}}_{\mathsf{SAN}}(m, m', i) = \bigwedge_{j=1}^{n} \big((m_j = m'_j) \vee a_{i,j}\big)$ — all modifications are allowed for sanitizer i,
- $\mathsf{ADM}^{\mathbf{A}}_{\mathsf{SET}}(m, m', \mathbf{Z}) = \bigwedge_{j=1}^{n} \big((m_j = m'_j) \vee (\exists i \in \mathbf{Z} \mid a_{i,j})\big)$ — each change is permitted for some sanitizer in $\mathbf{Z}$.

We also define an algorithm $\mathbf{A} \leftarrow \mathsf{ExtA}(\mathbf{SKZ}, \sigma)$ that derives the admissibility matrix from a signature σ using the sanitizers' secret keys $\mathbf{SKZ}$. We can also call the algorithm using one sanitizer's secret key to get his admissible blocks.

Our security experiments involve several oracles accessible to the adversary, fully detailed in Figure 1. Throughout the experiments, the challenger maintains a set Σ containing all fresh or sanitized signatures produced by the oracles. We provide a high level description of each of the oracles below:

$\mathcal{O}_{\mathsf{Sign}}$**:** Returns signatures generated using the signer's secret key held by the challenger. Σ is updated with the new signature.

$\mathcal{O}_{\mathsf{Sanit}}$**:** Returns sanitized signatures using one of the challenger's sanitizers' keys, provided the input signature is valid and the requested modification is admissible. Σ is updated with the new signature. The adversary chooses the sanitizer to do the operation by sending thehis public key to the challenger.

$\mathcal{O}_{\mathsf{Sanit}'}$**:** Behaves like $\mathcal{O}_{\mathsf{Sanit}}$, but with conditional history tracking. If the signer's public key differs from that of the challenger, sanitization is performed without maintaining history in Σ. Otherwise, history is maintained only if the original signature is already in Σ and the modification is admissible.

$\mathcal{O}^{b}_{\mathsf{LoRADM}}$**:** Takes a message and two admissibility matrices (A_0, A_1), and returns a signature generated using matrix A_b and the signer's secret key held by the challenger.

$\mathcal{O}^{b}_{\mathsf{LoRSanit}}$**:** Takes two sets of sanitizers, messages, modifications, and signatures. If both signatures are valid, the admissibility matrices are equal, the modifications are admissible, and the resulting modified messages are equal, the oracle returns a sanitized signature corresponding to a bit b. The signer's and sanitizers' secret keys are held by the challenger.

$\mathcal{O}^{b}_{\mathsf{LoRSanitPriv}}$**:** Takes as input two sets of sanitizers, messages, modifications, and an admissibility matrix. The oracle checks if both modifications produce the

same final message, if not it returns $\perp$, otherwise, it generates signatures on both messages and sanitizes the signatures with the corresponding modifications. If both signatures are valid, the oracle returns one of them according to a bit b. The operations are done using the signer's and sanitizers' secret keys which are held by the challenger.

$\mathcal{O}^{b}_{\mathsf{Sign/Sanit}}$**:** Takes a message, a modification and a sanitizer, and either signs the modified message directly or signs the original message and applies the modification via sanitization, depending on a bit b. The signer's and sanitizers' secret keys are held by the challenger. The history is kept in Σ' instead of Σ.

$\mathcal{O}_{\mathsf{Prove}}$**:** Allows the adversary to generate a proof on a given signature using the Prove algorithm which is called using the singer's secret key held by the challenger.

We now describe the security properties adapted for k-SAN.

Immutability. It ensures that a malicious sanitizer cannot modify inadmissible blocks. Formally, given access to the oracles $\{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Prove}}\}$, an adversary should not be able to produce a forgery $(m^*, \sigma^*, \mathbf{PKZ}^*)$ such that m^* results from inadmissible modifications under the admissibility matrix associated with $\mathbf{PKZ}^*$.

Definition 2 (Immutability [11]). *A k-sanitizer sanitizable signature scheme* k-SAN *is said to be immutable if for all $k \geq 1$ and* PPT *adversaries $\mathcal{A}$,* $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Immut},k}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ *where* $\mathsf{Exp}^{\mathsf{Immut},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$ *is defined in Figure 2.*

Accountability. It ensures that neither a dishonest signer nor sanitizer can falsely accuse the other party of generating a signature, *i.e.,* the Judge algorithm must not incorrectly return S (signer) or Z (sanitizer). This property is captured by two games. In the *signer-accountability* game, the adversary is given access to the oracle $\mathcal{O}_{\mathsf{Sanit}}$. It must produce a valid signature and proof $(\mathsf{pk}^*_{\mathsf{S}}, m^*, \sigma^*, \pi^*)$ where the signature was not obtained through this oracle, but Judge outputs Z on the associated π^*. Conversely, in the *sanitizer-accountability* game, the adversary is granted access to $\mathcal{O}_{\mathsf{Sign}}$ and $\mathcal{O}_{\mathsf{Prove}}$. Its goal is to produce a valid signature $(\mathbf{PKZ}^*, m^*, \sigma^*)$ not generated via these oracles, yet for which Judge outputs S.

Definition 3 (Sanitizer-Accountability [11]). *A k-sanitizer sanitizable signature scheme* k-SAN *is said to be sanitizer-accountable if for all* PPT *adversaries $\mathcal{A}$,* $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{SanAcc}}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ *where* $\mathsf{Exp}^{\mathsf{SanAcc}}_{\text{k-SAN},\mathcal{A}}(\lambda)$ *is defined in Figure 2.*

Definition 4 (Signer-Accountability [11]). *A k-sanitizer sanitizable signature scheme* k-SAN *is said to be signer-accountable if for all $k \geq 1$ and* PPT *adversaries $\mathcal{A}$,* $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ *where* $\mathsf{Exp}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$ *is defined in Figure 2.*

$\mathcal{O}_{\mathsf{Sign}}(\mathbf{PKZ}, m, \mathbf{A})$

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_\mathsf{S}^\dagger, \mathbf{PKZ}, m, \mathbf{A})$
$\Sigma := \Sigma \cup \{(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}, m, \mathbf{A}, \sigma)\}$
return σ

$\mathcal{O}_{\mathsf{Sanit}'}(\mathsf{pk}_\mathsf{Z}^{\dagger,i}, \mathsf{pk}_\mathsf{S}, m, \mathsf{MOD}, \sigma)$

if $\mathsf{pk}_\mathsf{S} \neq \mathsf{pk}_\mathsf{S}^\dagger$ **then**
  $\mathbf{A} \leftarrow \mathsf{ExtA}(\mathbf{SKZ}^\dagger, \sigma)$
  if $\neg \mathsf{ADM}_{\mathsf{SAN}}^{\mathbf{A}}(m, \mathsf{MOD}(m), i)$ **then**
    return $\perp$
  $\sigma' \leftarrow \mathsf{Sanitize}(\mathsf{sk}_\mathsf{Z}^{\dagger,i}, \mathsf{pk}_\mathsf{S}, \mathbf{PKZ}^\dagger, m, \mathsf{MOD}, \sigma)$
  return σ'
elseif $\begin{pmatrix} \exists(m', \sigma', \mathbf{A}') \in \Sigma \mid \\ m' = m \wedge \sigma' = \sigma \wedge \\ \mathsf{ADM}_{\mathsf{SAN}}^{\mathbf{A}'}(m, \mathsf{MOD}(m), i) \end{pmatrix}$ **then**
  $\sigma' \leftarrow \mathsf{Sanitize}(\mathsf{sk}_\mathsf{Z}^{\dagger,i}, \mathsf{pk}_\mathsf{S}, \mathbf{PKZ}^\dagger, m, \mathsf{MOD}, \sigma)$
  $\Sigma := \Sigma \cup \{(\mathsf{pk}_\mathsf{S}, \mathbf{PKZ}^\dagger, \mathsf{MOD}(m), \mathbf{A}', \sigma')\}$
  return σ'
return $\perp$

$\mathcal{O}_{\mathsf{LoRADM}}^b(\mathbf{PKZ}, m, \mathbf{A}_0, \mathbf{A}_1)$

if $\begin{cases} |\mathbf{A}_0| = |\mathbf{A}_1| = |\mathbf{PKZ}| \times |m| \\ \mathbf{PKZ} = \mathbf{PKZ}^\dagger \vee \mathbf{A}_0 = \mathbf{A}_1 \end{cases}$ **then**
  $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_\mathsf{S}^\dagger, \mathbf{PKZ}, m, \mathbf{A}_b)$
  if $\mathbf{PKZ} = \mathbf{PKZ}^\dagger$ **then**
    $\Sigma := \Sigma \cup \{(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}, m, \mathbf{A}_0 \wedge \mathbf{A}_1, \sigma)\}$
  return σ
return $\perp$

$\mathcal{O}_{\mathsf{Sign/Sanit}}^b(\mathsf{pk}_\mathsf{Z}^{\dagger,i}, m, \mathsf{MOD}, \mathbf{A})$

$m' := \mathsf{MOD}(m)$
if $\neg \mathsf{ADM}_{\mathsf{SAN}}^{\mathbf{A}}(m, m', i)$ **then return** $\perp$
if $b = 0$ **then**
  $\sigma' \leftarrow \mathsf{Sign}(\mathsf{sk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m', \mathbf{A})$
else
  $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m, \mathbf{A})$
  $\sigma' \leftarrow \mathsf{Sanitize}(\mathsf{sk}_\mathsf{Z}^{\dagger,i}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m, \mathsf{MOD}, \sigma)$
$\Sigma' := \Sigma' \cup \{(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m', \mathbf{A}, \sigma')\}$
return σ'

$\mathcal{O}_{\mathsf{Sanit}}(\mathsf{pk}_\mathsf{Z}^{\dagger,i}, \mathsf{pk}_\mathsf{S}, m, \mathsf{MOD}, \sigma)$

$\mathbf{A} \leftarrow \mathsf{ExtA}(\mathbf{SKZ}^\dagger, \sigma)$
if $\begin{cases} \mathsf{Verify}(\mathsf{pk}_\mathsf{S}, \mathbf{PKZ}^\dagger, m, \sigma) \\ \mathsf{ADM}_{\mathsf{SAN}}^{\mathbf{A}}(m, \mathsf{MOD}(m), i) \end{cases}$ **then**
  $\sigma' \leftarrow \mathsf{Sanitize}(\mathsf{sk}_\mathsf{Z}^{\dagger,i}, \mathsf{pk}_\mathsf{S}, \mathbf{PKZ}^\dagger, m, \mathsf{MOD}, \sigma)$
  $\Sigma := \Sigma \cup \{(\mathsf{pk}_\mathsf{S}, \mathbf{PKZ}^\dagger, \mathsf{MOD}(m), \mathbf{A}, \sigma')\}$
  return σ'
return $\perp$

$\mathcal{O}_{\mathsf{LoRSanit}}^b\begin{pmatrix} \mathsf{pk}_\mathsf{Z}^{\dagger,i_0}, m_0, \mathsf{MOD}_0, \sigma_0, \\ \mathsf{pk}_\mathsf{Z}^{\dagger,i_1}, m_1, \mathsf{MOD}_1, \sigma_1 \end{pmatrix}$

$b_\beta \leftarrow \mathsf{Verify}(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m_\beta, \sigma_\beta), \forall \beta \in \{0,1\}$
if $\neg b_0 \vee \neg b_1$ **then return** $\perp$
foreach $\beta \in \{0,1\}$ **do**
  $\mathbf{A}_\beta \leftarrow \mathsf{ExtA}(\mathsf{sk}_\mathsf{Z}^{\dagger,i_\beta}, \sigma_\beta)$
  $\sigma'_\beta \leftarrow \mathsf{Sanitize}(\mathsf{sk}_\mathsf{Z}^{\dagger,i_\beta}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m_\beta, \mathsf{MOD}_\beta, \sigma_\beta)$
if $\begin{cases} \mathbf{A}_0 = \mathbf{A}_1 \\ \mathsf{ADM}_{\mathsf{SAN}}^{\mathbf{A}_0}(m_0, \mathsf{MOD}(m_0), i_0) \\ \mathsf{ADM}_{\mathsf{SAN}}^{\mathbf{A}_1}(m_1, \mathsf{MOD}(m_1), i_1) \\ \mathsf{MOD}(m_0) = \mathsf{MOD}(m_1) \end{cases}$ **then**
  $\Sigma := \Sigma \cup \{(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, \mathsf{MOD}_b(m_b), \mathbf{A}_b, \sigma'_b)\}$
  return σ'_b
return $\perp$

$\mathcal{O}_{\mathsf{Prove}}(\mathbf{PKZ}, m, \sigma, j)$

if $\mathbf{PKZ} = \mathbf{PKZ}^\dagger \wedge (m, \sigma) \in \Sigma'$ **then return** $\perp$
if $\mathsf{Verify}(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}, m, \sigma) = 0$ **then return** $\perp$
$\pi \leftarrow \mathsf{Prove}(\mathsf{sk}_\mathsf{S}^\dagger, \mathbf{PKZ}, m, \sigma, j)$
return π

$\mathcal{O}_{\mathsf{LoRSanitPriv}}^b\begin{pmatrix} \mathsf{pk}_\mathsf{Z}^{\dagger,i_0}, m_0, \mathsf{MOD}_0, \\ \mathsf{pk}_\mathsf{Z}^{\dagger,i_1}, m_1, \mathsf{MOD}_1, \mathbf{A} \end{pmatrix}$

if $\mathsf{MOD}_0(m_0) \neq \mathsf{MOD}_1(m_1)$ **then return** $\perp$
foreach $\beta \in \{0,1\}$ **do**
  $\sigma_\beta \leftarrow \mathsf{Sign}(\mathsf{sk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m_\beta, \mathbf{A})$
  $\sigma'_\beta \leftarrow \mathsf{Sanitize}(\mathsf{sk}_\mathsf{Z}^{\dagger,i_\beta}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m_\beta, \mathsf{MOD}_\beta, \sigma_\beta)$
  $b_\beta \leftarrow \mathsf{Verify}(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m_\beta, \sigma'_\beta)$
  if $b_\beta \neq 1$ **then return** $\perp$
return σ'_β

Fig. 1: Oracles for k-SAN. $\mathsf{pk}_\mathsf{S}^\dagger$, $\mathsf{sk}_\mathsf{S}^\dagger$, $\mathbf{PKZ}^\dagger$, and $\mathbf{SKZ}^\dagger$ are the keys generated and kept by the challenger. $\mathsf{pk}_\mathsf{Z}^{\dagger,i}$ denotes the public key of the sanitizer that the adversary wants to use to call the $\mathsf{Sanitize}$ algorithm within the oracles. Upon receiving a $\mathsf{pk}_\mathsf{Z}^{\dagger,i}$, the oracles search directly for the corresponding secret key $\mathsf{sk}_\mathsf{Z}^{\dagger,i}$ and index i in $\mathbf{SKZ}^\dagger$ and keep them in memory.

$\mathsf{KGenAll}(k)$

$\mathbf{SKZ}^\dagger := \bot, \mathbf{PKZ}^\dagger := \bot, \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
$(\mathsf{sk}_\mathsf{S}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger) \leftarrow \mathsf{KGen}_\mathsf{S}(\mathsf{pp})$
foreach $i \in [\![k]\!]$ **do**
  $(\mathsf{sk}_\mathsf{Z}^i, \mathsf{pk}_\mathsf{Z}^i) \leftarrow \mathsf{KGen}_\mathsf{Z}(\mathsf{pp})$
  $\mathbf{SKZ}^\dagger \leftarrow \mathsf{Append}(\mathbf{SKZ}^\dagger, \mathsf{sk}_\mathsf{Z}^i)$
  $\mathbf{PKZ}^\dagger \leftarrow \mathsf{Append}(\mathbf{PKZ}^\dagger, \mathsf{pk}_\mathsf{Z}^i)$
return $(\mathsf{pp}, \mathsf{sk}_\mathsf{S}^\dagger, \mathbf{SKZ}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger)$

$\mathsf{Exp}^{\mathsf{Unforg},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$

$\Sigma := \emptyset, \mathbb{O} \leftarrow \{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}_{\mathsf{Prove}}\}$
$(\mathsf{pp}, \mathsf{sk}_\mathsf{S}^\dagger, \mathbf{SKZ}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger) \leftarrow \mathsf{KGenAll}(k)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{pp}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger)$
$b_0 \leftarrow \mathsf{Verify}(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m^*, \sigma^*)$
$b_1 := ((\mathsf{pk}_\mathsf{S}^\dagger, m^*, \sigma^*) \notin \{(\mathsf{pk}_\mathsf{S}^i, m_i, \sigma_i)\}_{i=1}^{|\Sigma|})$
return $b_0 \wedge b_1$

$\mathsf{Exp}^{\mathsf{Unlink},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$

$\Sigma := \emptyset, \mathbb{O} \leftarrow \{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}^b_{\mathsf{LoRSanit}}, \mathcal{O}_{\mathsf{Prove}}\}$
$(\mathsf{pp}, \mathsf{sk}_\mathsf{S}^\dagger, \mathbf{SKZ}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger) \leftarrow \mathsf{KGenAll}(k)$
$b^* \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{pp}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger)$
return $b = b^*$

$\mathsf{Exp}^{\mathsf{FullSanit},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$

$\Sigma := \emptyset$
$(\mathsf{pp}, \mathsf{sk}_\mathsf{S}^\dagger, \mathbf{SKZ}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger) \leftarrow \mathsf{KGenAll}(k)$
$(\mathsf{pk}_\mathsf{S}^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sanit}}}(\mathsf{pp}, \mathbf{PKZ}^\dagger)$
$b_0 \leftarrow \mathsf{Verify}(\mathsf{pk}_\mathsf{S}^*, \mathbf{PKZ}^\dagger, m^*, \sigma^*)$
$b_1 := \exists j \in [\![n]\!], \sigma^*.\mathbf{PA}_j = 1$
$b_2 := ((\mathsf{pk}_\mathsf{S}^*, m^*, \sigma^*) \notin \{(\mathsf{pk}_\mathsf{S}^i, m_i, \sigma_i)\}_{i=1}^{|\Sigma|})$
return $b_0 \wedge b_1 \wedge b_2$

$\mathsf{Exp}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$

$\Sigma := \emptyset$
$(\mathsf{pp}, \mathsf{sk}_\mathsf{S}^\dagger, \mathbf{SKZ}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger) \leftarrow \mathsf{KGenAll}(k)$
$(\mathsf{pk}_\mathsf{S}^*, m^*, \sigma^*, \pi^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sanit}}}(\mathsf{pp}, \mathbf{PKZ}^\dagger)$
$b_0 \leftarrow \mathsf{Verify}(\mathsf{pk}_\mathsf{S}^*, \mathbf{PKZ}^\dagger, m^*, \sigma^*)$
$b_1 := ((\mathsf{pk}_\mathsf{S}^*, m^*, \sigma^*) \notin \{(\mathsf{pk}_\mathsf{S}^i, m_i, \sigma_i)\}_{i=1}^{|\Sigma|})$
$b_2 := \mathsf{Judge}(\mathsf{pk}_\mathsf{S}^*, \mathbf{PKZ}^\dagger, m^*, \sigma^*, \pi^*, \bot) \neq S$
return $b_0 \wedge b_1 \wedge b_2$

$\mathsf{Exp}^{\mathsf{AdmSec},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$

$(\mathsf{pp}, \mathsf{sk}_\mathsf{S}^\dagger, \mathbf{SKZ}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger) \leftarrow \mathsf{KGenAll}(k)$
$\mathbb{O} := \{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Prove}}\}$
$(m, i', j') \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{pp}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger)$
foreach $i \in [\![k]\!], j \in [\![n]\!]$ **do**
  $a_{i,j} := \begin{cases} 1, j = j' \wedge i = i' \wedge b = 1 \\ 0, \text{otherwise} \end{cases}$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m, \mathbf{A})$
$b^* \leftarrow \mathcal{A}^{\mathbb{O}}(\sigma)$
return $b = b^*$

$\mathsf{Exp}^{\mathsf{Immut},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$

$\Sigma := \emptyset, (\mathsf{pp}, \mathsf{sk}_\mathsf{S}^\dagger, \mathbf{SKZ}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger) \leftarrow \mathsf{KGenAll}(k)$
$(\mathbf{PKZ}^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Prove}}}(\mathsf{pp}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger)$
$b_0 \leftarrow \mathsf{Verify}(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^*, m^*, \sigma^*), b_1 := 0$
foreach $(\mathbf{PKZ}_i, m_i, \mathbf{A}_i, \sigma_i) \in \Sigma$ **do**
  $\mathbf{Z} := \{l \in [1, |\mathbf{PKZ}_i|] \mid \mathsf{pk}_\mathsf{Z}^l \notin \mathbf{PKZ}^\dagger\}$
  if $\begin{cases} \mathbf{PKZ}^* = \mathbf{PKZ}_i \\ \mathsf{ADM}^{\mathbf{A}_i}_{\mathsf{SET}}(m_i, m^*, \mathbf{Z}) \end{cases}$
    $b_1 := 1$
return $b_0 \wedge \neg b_1$

$\mathsf{Exp}^{\mathsf{Trans},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$

$\Sigma := \emptyset, \mathbb{O} \leftarrow \{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}^b_{\mathsf{Sign/Sanit}}, \mathcal{O}_{\mathsf{Prove}}\}$
$(\mathsf{pp}, \mathsf{sk}_\mathsf{S}^\dagger, \mathbf{SKZ}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger) \leftarrow \mathsf{KGenAll}(k)$
$b^* \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{pp}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger)$
return $b = b^*$

$\mathsf{Exp}^{\mathsf{Inv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$

$\Sigma := \emptyset, \mathbb{O} \leftarrow \{\mathcal{O}_{\mathsf{Sanit}'}, \mathcal{O}^b_{\mathsf{LoRADM}}, \mathcal{O}_{\mathsf{Prove}}\}$
$(\mathsf{pp}, \mathsf{sk}_\mathsf{S}^\dagger, \mathbf{SKZ}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger) \leftarrow \mathsf{KGenAll}(k)$
$b^* \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{pp}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger)$
return $b = b^*$

$\mathsf{Exp}^{\mathsf{Priv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$

$\Sigma := \emptyset$
$\mathbb{O} \leftarrow \{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}^b_{\mathsf{LoRSanitPriv}}, \mathcal{O}_{\mathsf{Prove}}\}$
$(\mathsf{pp}, \mathsf{sk}_\mathsf{S}^\dagger, \mathbf{SKZ}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger) \leftarrow \mathsf{KGenAll}(k)$
$b^* \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{pp}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger)$
return $b = b^*$

$\mathsf{Exp}^{\mathsf{SanAcc}}_{\text{k-SAN},\mathcal{A}}(\lambda)$

$\Sigma := \emptyset, \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda), (\mathsf{sk}_\mathsf{S}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger) \leftarrow \mathsf{KGen}_\mathsf{S}(\mathsf{pp})$
$(\mathbf{PKZ}^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Prove}}}(\mathsf{pp}, \mathsf{pk}_\mathsf{S}^\dagger)$
$\pi^* \leftarrow \mathsf{Prove}(\mathsf{sk}_\mathsf{S}^\dagger, \mathbf{PKZ}^*, m^*, \sigma^*, \bot)$
$b_0 \leftarrow \mathsf{Verify}(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^*, m^*, \sigma^*)$
$b_1 := ((\mathbf{PKZ}^*, m^*, \sigma^*) \notin \{((\mathbf{PKZ}_i, m_i, \sigma_i))\}_{i=1}^{|\Sigma|})$
$b_2 := (\mathsf{Judge}(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^*, m^*, \sigma^*, \pi^*, \bot) \neq Z)$
return $b_0 \wedge b_1 \wedge b_2$

$\mathsf{Exp}^{\mathsf{SanitAnon},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$

$(\mathsf{pp}, \mathsf{sk}_\mathsf{S}^\dagger, \mathbf{SKZ}^\dagger, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger) \leftarrow \mathsf{KGenAll}(k)$
$\mathbb{O} := \{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}_{\mathsf{Prove}}\}$
$(m, i_0, i_1, j', m'_{j'}) \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{pp}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger)$
foreach $i \in [\![k]\!], j \in [\![n]\!]$ **do**
  $a_{i,j} := \begin{cases} 1, j = j' \wedge i \in \{i_0, i_1\} \\ 0, \text{otherwise} \end{cases}$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m, \mathbf{A})$
$\sigma' \leftarrow \mathsf{Sanitize}(\mathsf{sk}_\mathsf{Z}^{\dagger, i_b}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m, (j, m'_{j'}), \sigma)$
$b^* \leftarrow \mathcal{A}^{\mathbb{O}}(\sigma')$
return $b = b^*$

Fig. 2: Security Experiments for k-SAN.

Unforgeability. It requires that, given access to the oracles $\{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}_{\mathsf{Prove}}\}$, an adversary cannot produce a valid message and signature (m^*, σ^*) such that σ^* was not generated by the oracles.

Definition 5 (Unforgeability [2]). *A k-sanitizer sanitizable signature scheme* k-SAN *is said to be unforgeable if for all $k \geq 1$ and* PPT *adversaries $\mathcal{A}$,* $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Unforg},k}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ *where* $\mathsf{Exp}^{\mathsf{Unforg},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$ *is defined in Figure 2.*

As established in [6, 24], in sanitizable signatures, having both *signer accountability* and *sanitizer accountability* implies *unforgeability*. This result extends naturally to the k-Sanitizer setting. Indeed, a non-negligible advantage against our *unforgeability* experiment implies that the adversary can output a new valid message-signature pair (m^*, σ^*) (*i.e.,* $b_0 = b_1 = 1$ in the experiment). This pair also satisfies the same validity conditions in $\mathsf{Exp}^{\mathsf{SanAcc}}_{\text{k-SAN},\mathcal{A}}(\lambda)$ and $\mathsf{Exp}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$. On input (m^*, σ^*), the Judge algorithm must output either S or Z, thus breaking either *Signer Accountability* or *Sanitizer Accountability* with non-negligible probability.

Proof-Restricted Transparency. Transparency requires that sanitized signatures are indistinguishable from fresh signatures. Given a signature on a message m that was sanitized if $b = 1$ and not sanitized if $b = 0$, the adversary should not be able to guess b with probability significantly better than random guessing. However, this property cannot hold if the adversary has access to a Prove oracle that can distinguish sanitized from fresh signatures. Therefore, we consider a relaxed notion called *proof-restricted transparency*, which ensures that the adversary cannot win without using the prove oracle. In this setting, the adversary is given access to the oracles $\{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}^b_{\mathsf{Sign/Sanit}}, \mathcal{O}_{\mathsf{Prove}}\}$, with the restriction that $\mathcal{O}_{\mathsf{Prove}}$ returns $\perp$ on signatures generated via $\mathcal{O}^b_{\mathsf{Sign/Sanit}}$.

Definition 6 (Proof-Restricted Transparency [11]). *A k-sanitizer sanitizable signature scheme* k-SAN *is said to be proof-restricted transparent if for all $k \geq 1$ and* PPT *adversaries $\mathcal{A}$,* $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Trans},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ *where* $\mathsf{Exp}^{\mathsf{Trans},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$ *is defined in Figure 2.*

Privacy. Privacy requires that sanitized signatures do not leak information about the original message. The privacy game follows a left-or-right indistinguishability approach: the adversary is given access to the oracle $\mathcal{O}^b_{\mathsf{LoRSanitPriv}}$ which returns a sanitized signature that was produced by one of two modifications resulting in the same final message according to a random bit b. The adversary should not be able to guess $b^* = b$ with probability significantly better than random guessing. The adversary is also given access to the oracles $\{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}_{\mathsf{Prove}}\}$.

Definition 7 (Privacy [2]). *A k-sanitizer sanitizable signature scheme* k-SAN *is said to be private if for all* $k \geq 1$ *and* PPT *adversaries* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Priv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ *where* $\mathsf{Exp}^{\mathsf{Priv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$ *is defined in Figure 2.*

Proof-restricted transparency implies privacy as explained in [6, 24] which also applies to our k-SAN model.

Invisibility. It ensures that an external observer cannot determine whether a specific block is admissible, nor which sanitizer is authorized to modify it. The invisibility game follows a left-or-right indistinguishability approach: given a signature associated with an admissibility matrix $\mathbf{A}_b$, chosen at random between $\mathbf{A}_0$ and $\mathbf{A}_1$, the adversary must guess b. The adversary wins if they output $b^* = b$ with probability significantly better than random guessing while having access to the oracles $\{\mathcal{O}_{\mathsf{Sanit}'}, \mathcal{O}^b_{\mathsf{LoRADM}}, \mathcal{O}_{\mathsf{Prove}}\}$.

Definition 8 (Invisibility [11]). *A k-sanitizer sanitizable signature scheme* k-SAN *is said to be invisible if for all* $k \geq 1$ *and* PPT *adversaries* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Inv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ *where* $\mathsf{Exp}^{\mathsf{Inv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$ *is defined in Figure 2.*

Unlinkability. It ensures that an external observer cannot determine the origin of a sanitized signature. The game follows a left-or-right indistinguishability approach: given a sanitized signature σ'_b derived from either σ_0 or σ_1, the adversary must guess $b^* = b$ with probability significantly better than random guessing given access to the oracles $\{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}^b_{\mathsf{LoRSanit}}, \mathcal{O}_{\mathsf{Prove}}\}$ in order to win.

Definition 9 (Unlinkability [11]). *A k-sanitizer sanitizable signature scheme* k-SAN *is said to be unlinkable if for all* $k \geq 1$ *and* PPT *adversaries* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Unlink},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ *where* $\mathsf{Exp}^{\mathsf{Unlink},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$ *is defined in Figure 2.*

Sanitizer Anonymity. It requires that the identity of the sanitizer who modified an admissible block remains hidden. Given k sanitizers in **PKZ**, the adversary selects sanitizers (i_0, i_1), a message m, and a modified block $m'_{j'}$. The challenger signs m and sets both i_0 and i_1 as authorized for block j', then picks $b \in \{0, 1\}$ and performs sanitization with sanitizer i_b. The adversary wins if it can guess b with a probability significantly better than random guessing. The adversary has access to $\{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}_{\mathsf{Prove}}\}$.

In case only one sanitizer is authorized, anonymity relies on the admissibility matrix secrecy. The adversary chooses a sanitizer i', m, and block j'; the challenger randomly sets i' as authorized for j' iff $b = 1$. The adversary must guess b with a probability significantly better than random guessing given access to $\{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Prove}}\}$.

Definition 10 (Sanitizer Anonymity [14]). *A k-sanitizer sanitizable signature scheme* k-SAN *is said to be sanitizer anonymous if for all $k \geq 2$ and* PPT *adversaries $\mathcal{A}$,* $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{SanitAnon},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1 \vee \mathsf{Exp}^{\mathsf{AdmSec},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ *where the experiments $\mathsf{Exp}^{\mathsf{SanitAnon},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$ and $\mathsf{Exp}^{\mathsf{AdmSec},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$ are defined in Figure 2.*

Full-Sanitization Verifiability. Full-sanitization verifiability requires that a signature is valid if and only if all admissible blocks were sanitized. In the game, the adversary is given a list of sanitizer public keys $\mathbf{PKZ}^{\dagger}$ and access to the oracle $\mathcal{O}_{\mathsf{Sanit}}$. He should produce a signature $(\mathsf{pk}^{*}_{\mathsf{S}}, m^{*}, \sigma^{*})$ where σ^{*} is valid under $\mathbf{PKZ}^{\dagger}$ with at least one admissible block according to a public admissibility vector $\sigma^{*}.\mathbf{PA}$ provided in the signature. In addition, the signature should not have been produced using $\mathcal{O}_{\mathsf{Sanit}}$.

Definition 11 (Full-Sanitization Verifiability). *A k-sanitizer sanitizable signature scheme* k-SAN *is said to be full-sanitization verifiable if for all $k \geq 1$ and* PPT *adversaries $\mathcal{A}$,* $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{FullSanit},k}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ *where* $\mathsf{Exp}^{\mathsf{FullSanit},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$ *is defined in Figure 2.*

Full-sanitization verifiability breaks proof-restricted transparency and invisibility which we prove below.

Theorem 1. *If a* k-SAN *construction is full-sanitization verifiable, then it does not satisfy the proof-restricted transparency property.*

Proof. Assume, for contradiction, that there exists a k-SAN construction that is both full-sanitization verifiable and proof-restricted transparent. Let $\mathcal{A}$ be an adversary in the $\mathsf{Exp}^{\mathsf{Trans},k,b}_{\text{k-SAN},\mathcal{A}}$ game which we construct as the following. Firstly, $\mathcal{A}$ receives the challenge $(\mathsf{pp}, \mathsf{pk}^{\dagger}_{\mathsf{S}}, \mathbf{PKZ}^{\dagger})$ from the proof-restricted transparency challenger. Then, $\mathcal{A}$ picks a message block j and a sanitizer i at random. $\mathcal{A}$ sets the admissibility matrix $\mathbf{A}$ such that $a_{i,j} = 1$ and all other entries are 0. Then, $\mathcal{A}$ sets the message m such that all blocks have the value 1 and sets $\mathsf{MOD} = ((j, 0))$. Next, $\mathcal{A}$ calls the $\mathcal{O}^{b}_{\mathsf{Sign}/\mathsf{Sanit}}(\mathsf{pk}^{\dagger,i}_{\mathsf{Z}}, m, \mathsf{MOD}, \mathbf{A})$ oracle which will return a signature σ. Finally, $\mathcal{A}$ calls $\mathsf{Verify}(\mathsf{pk}^{\dagger}_{\mathsf{S}}, \mathbf{PKZ}^{\dagger}, m, \sigma)$ and gets b. If $b = 1$, $\mathcal{A}$ returns $b^{*} = 1$, otherwise it returns $b^{*} = 0$. By the definition of full-sanitization verifiability, σ is valid if and only if the $\mathsf{Sanitize}$ algorithm was called on the admissible message block j. Therefore, $\mathcal{A}$ can detect sanitization, giving it a non-negligible advantage in the proof-restricted transparency game. This contradicts the definition of proof-restricted transparency. Thus, no k-SAN construction can satisfy both properties simultaneously. □

Theorem 2. *If a* k-SAN *construction is full-sanitization verifiable, then it does not satisfy the invisibility property.*

Proof. Assume, for contradiction, that there exists a k-SAN construction that is both full-sanitization verifiable and invisible. Let $\mathcal{A}$ be an adversary in the $\mathsf{Exp}^{\mathsf{Inv},k,b}_{\text{k-SAN},\mathcal{A}}$ game which we construct as the following. Firstly, $\mathcal{A}$ receives the challenge $(\mathsf{pp}, \mathsf{pk}^{\dagger}_{\mathsf{S}}, \mathbf{PKZ}^{\dagger})$ from the invisibility challenger. Then, $\mathcal{A}$ picks a message block j and a sanitizer i at random. $\mathcal{A}$ sets the admissibility matrix $\mathbf{A}_0$ such that $a_{i,j} = 1$ and all other entries are 0, and sets all the entries of the admissibility matrix $\mathbf{A}_1$ as 0. Then, $\mathcal{A}$ sets the message m such that all blocks have the value 1 and calls the $\mathcal{O}^{b}_{\mathsf{LoRADM}}(\mathbf{PKZ}^{\dagger}, m, \mathbf{A}_0, \mathbf{A}_1)$ oracle which will return a signature σ. Finally, $\mathcal{A}$ calls $\mathsf{Verify}(\mathsf{pk}^{\dagger}_{\mathsf{S}}, \mathbf{PKZ}^{\dagger}, m, \sigma)$ and gets b. If $b = 1$, $\mathcal{A}$ returns $b^* = 1$, otherwise it returns $b^* = 0$. By the definition of full-sanitization verifiability, σ is valid if and only if there are no admissible blocks (*i.e.,* $\mathbf{A}_1$ was used to generate σ) as the $\mathcal{O}^{b}_{\mathsf{LoRADM}}$ oracle does not call the $\mathsf{Sanitize}$ algorithm. Therefore, $\mathcal{A}$ can distinguish whether $\mathbf{A}_0$ or $\mathbf{A}_1$ was used to generate σ, giving it a non-negligible advantage in the invisibility game. This contradicts the definition of invisibility. Thus, no k-SAN construction can satisfy both properties simultaneously. □

4 Cryptographic Building Blocks

We now introduce the building blocks required by our constructions informally whereas the formal definitions of the algorithms and security can be found in Appendices A and B.

Chameleon Hash functions which were introduced in [23] are commonly used in sanitizable signatures. They are described as CHash := ($\mathsf{ParGen}_{\mathsf{CHash}}$, $\mathsf{KGen}_{\mathsf{CHash}}$, $\mathsf{Hash}_{\mathsf{CHash}}$, $\mathsf{Check}_{\mathsf{CHash}}$, $\mathsf{Adapt}_{\mathsf{CHash}}$). They encompass a relaxed property of *collision resistance* compared to hash functions, allowing the holder of a trapdoor generated in $\mathsf{KGen}_{\mathsf{CHash}}$ to find collisions for a hash generated by $\mathsf{Hash}_{\mathsf{CHash}}$ using the algorithm $\mathsf{Adapt}_{\mathsf{CHash}}$. Without the trapdoor, they behave like normal hash functions. A CHash is secure if it is correct, collision resistant, unique, and indistinguishable.

We also use a *Digital Signature* scheme and a *Public Key Encryption* scheme described as SIG := ($\mathsf{KGen}_{\mathsf{SIG}}$, $\mathsf{Sign}_{\mathsf{SIG}}$, $\mathsf{Verify}_{\mathsf{SIG}}$) and PKE := ($\mathsf{KGen}_{\mathsf{PKE}}$, $\mathsf{Encrypt}_{\mathsf{PKE}}$, $\mathsf{Decrypt}_{\mathsf{PKE}}$, $\mathsf{Multiply}_{\mathsf{PKE}}$) respectively. PKE allows homomorphic scalar multiplication of the ciphertext using the $\mathsf{Multiply}_{\mathsf{PKE}}$ algorithm. SIG is considered secure if it is correct and has EUF-CMA security while PKE should be correct, IND-CPA secure, and *unlinkable* [28] which means that an adversary cannot predict whether the ciphertext was manipulated by $\mathsf{Multiply}_{\mathsf{PKE}}$ or generated freshly using $\mathsf{Encrypt}_{\mathsf{PKE}}$.

Our constructions also require a *Verifiable Ring Signature* described as VRS := ($\mathsf{Setup}_{\mathsf{VRS}}$, $\mathsf{KGen}_{\mathsf{VRS}}$, $\mathsf{Sign}_{\mathsf{VRS}}$, $\mathsf{Verify}_{\mathsf{VRS}}$, $\mathsf{Prove}_{\mathsf{VRS}}$, $\mathsf{Judge}_{\mathsf{VRS}}$). Ring Signatures allow members of a group of signers called a ring to sign messages anonymously within the ring, *i.e.,* an external entity cannot know which member of the ring signed the message [29]. Verifiable Ring Signatures (VRS) expand this concept

by allowing a member of the ring to prove that he is the signer of the message using the $\mathsf{Prove}_{\mathsf{VRS}}$ algorithm which can be verified later using the $\mathsf{Judge}_{\mathsf{VRS}}$ algorithm [26]. Bultel and Lafourcade [10] extended the definition of VRS to allow the member to prove also that he is not the signer. We require VRS to have the following security properties in order to be considered secure: correctness, unforgeability, anonymity, accountability which requires that an adversary cannot forge a signature along with a proof that he is not the signer, and non-seizability which requires that an adversary cannot accuse an honest user of signing a message.

Equivalence Class Signatures described as $\mathrm{EQS} := (\mathsf{BGGen}, \mathsf{KGen}_{\mathsf{EQS}}, \mathsf{Sign}_{\mathsf{EQS}}, \mathsf{ChgRep}_{\mathsf{EQS}}, \mathsf{Verify}_{\mathsf{EQS}}, \mathsf{VerifyKey}_{\mathsf{EQS}})$ allows signing representatives of equivalence classes using $\mathsf{Sign}_{\mathsf{EQS}}$, such that a representative and its corresponding signature can be adapted to give a fresh signature of a random representative in the same class [11] using $\mathsf{ChgRep}_{\mathsf{EQS}}$. The algorithm BGGen gives the description of a multiplicative bilinear group of prime order q which is defined in Appendix A. EQS is secure if it is correct, has EUF-CMA security, and has perfect-adaptation.

5 Invisible-Unlinkable-Transparent k-SAN

In the IUT-k-SAN construction, presented in Figure 3, the signer holds a pair of keys for EQS, while each sanitizer possesses a pair of PKE keys. Both entities also have key pairs for VRS.

During the Sign procedure, the signer generates n ephemeral signing key pairs following the BLS-like signature construction of [11]. Each key is used to sign a message block m_j after applying a hash function $\mathcal{H}(j\|m_j) \to \mathbb{G}_2$. These signatures are referred to as inner signatures. The signer then signs the verification keys of these inner signatures using his long-term EQS signing key, producing an outer signature. To enable sanitization, a ciphertext matrix $\mathbf{SKZ}$ is constructed by setting $\mathbf{SKZ}_{i,j}$ to an encryption of the signing key for block j under the public key of sanitizer i if it is admissible for him, and an encryption of 0 otherwise. Finally, all public information including the ciphertexts are signed using VRS over a ring containing the signer's and the k sanitizers' public keys.

To sanitize a message in the $\mathsf{Sanitize}$ algorithm, the sanitizer decrypts his ciphertexts in $\mathbf{SKZ}$ to recover the signing keys. For every block he wants to modify, the sanitizer re-signs the modified block using the inner signature scheme. To ensure unlinkability, all linkable elements in the signature are re-randomized. Specifically, BLS-like signatures and their secret and verification keys are re-randomized using fresh scalars r and s, yielding the transformation $(y, \sigma, (G_1^x, G_1^{xy})) \mapsto (y \cdot s, \sigma^s, (G_1^{rx}, G_1^{rsxy}))$. The outer signature must also be re-randomized to maintain unlinkability. This is achieved using the $\mathsf{ChgRep}_{\mathsf{EQS}}$ algorithm which re-randomizes signatures within the same equivalence class. Moreover, the ciphertexts in $\mathbf{SKZ}$ need to be re-randomized and adapted to accommodate the transformation of the signing key without access to the decryption keys of other sanitizers. This is done using the algorithm $\mathsf{Multiply}_{\mathsf{PKE}}$ which allows unlinkable

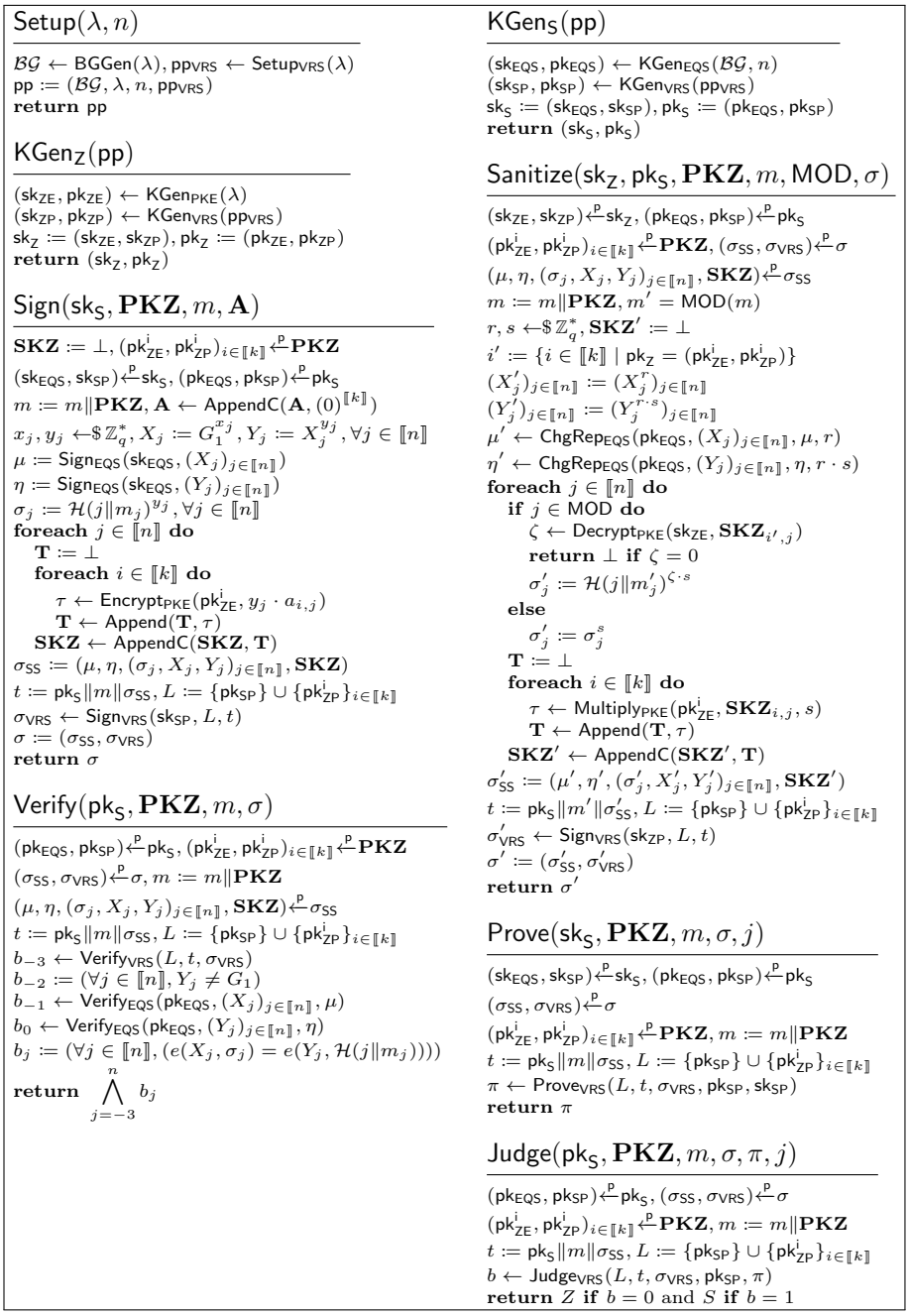

Fig. 3: Algorithms of the IUT-k-SAN construction.

homomorphic scalar multiplication. Finally, a new VRS signature is generated by the sanitizer on the modified public information.

In the Verify algorithm, correctness is verified by checking the EQS signatures, VRS signature, and by validating each inner signature over a message block j via the bilinear pairing equation: $e(G_1^{x_j}, \sigma) = e(G_1^{x_j y_j}, \mathcal{H}(j \| m_j))$.

The Prove algorithm uses the $\mathsf{Prove}_{\mathsf{VRS}}$ procedure to construct a proof of origin, while the Judge algorithm verifies the proof using $\mathsf{Judge}_{\mathsf{VRS}}$ which returns $b = 1$ if the signer created the signature, and $b = 0$ otherwise.

Since our construction does not achieve full-sanitization verifiability, the index j in the Prove and Judge algorithms is always set to $\perp$. While our security model does not require identifying the specific sanitizer who performed a modification, it is possible to extend the functionalities of this construction by allowing the identification of the exact sanitizer behind the VRS signature if all ring members generate proofs via the $\mathsf{Prove}_{\mathsf{VRS}}$ algorithm.

To create IUT-k-SAN, we used the construction of Bultel *et al.* [11] as a baseline, and then we did the necessary changes to satisfy the requirements of k-SAN. Notably, in [11], the inner signature keys are encrypted in a single ciphertext. During sanitization, after the keys are randomized, a new ciphertext is generated which ensures unlinkability. In the k-SAN context, we have the ciphertexts' matrix **SKZ** instead of a single ciphertext and during sanitization, the ciphertexts are updated using homomorphic operations to apply the randomization of the keys. We also extended the VRS ring to include the signer and all sanitizers. These changes along with the fact that the k-SAN security model is different from traditional sanitizable signatures require revisiting the security proofs.

Theorem 3. IUT-k-SAN *verifies correctness, unforgeability, immutability, accountability, privacy, proof-restricted transparency, invisibility, unlinkability, and sanitizer anonymity, if the underlying building blocks* PKE*,* EQS*,* VRS*, and BLS-like signatures are secure.*

Proof. The security proofs can be found in Appendix C. Here, we will explain the general idea of the proofs. We note that correctness is trivial as it relies directly on the correctness of the building blocks and that unforgeability and privacy are implied by accountability and proof-restricted transparency respectively. We use reduction based proof for the other properties.

Immutability. To prove this property we starts by using the unforgeability of BLS-like signature and the hardness of the problem:

Lemma 1. *Let* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G_1, G_2, G_T, e, q) \leftarrow \mathsf{BGGen}(\lambda)$ *with* $q > 2^\lambda$*, and* $a, b, c \leftarrow\!\!\$\, \mathbb{Z}_q$*. For all generic group adversary* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *on input* $(G_1, G_1^a, G_1^b, G_2, G_2^b, G_2^c)$ *outputs* $(G_1^u, G_1^v, G_1^x, G_1^y, G_2^z)$ *such that* $au - x = 0$*,* $bv - y = 0$*,* $cy - xz = 0$*, and* $v \neq 0$ *is negligible.*

The proof of this Lemma is provided in Appendix C.

Subsequently we show that the BLS-like verification keys cannot be manipulated assuming the EUF-CMA security of EQS. After that, the adversary's last mean of attack is to get the secret keys from the ciphertexts which, by the IND-CPA security of PKE, is intractable.

Accountability. The proof reduces the accountability games to those of VRS. Winning the signer-accountability game means the adversary produced a proof that they *did not* produce a valid VRS, thus violating VRS accountability. Likewise, winning the sanitizer-accountability game implies that the sanitizer created a VRS forgery for which the Prove algorithm falsely attributes the signature to the signer, thereby breaking the non-seizability of VRS.

Proof-Restricted Transparency. We prove, via a sequence of games, that the distribution of a fresh signature is indistinguishable from that of a sanitized one. This relies on the indistinguishability between several components. An adversary wins if he can:

1. distinguish whether the EQS signature was produced via $\mathsf{ChgRep}_{\mathsf{EQS}}$ or directly with $\mathsf{Sign}_{\mathsf{EQS}}$;
2. tell whether the ciphertexts in $\mathbf{SKZ}$ were generated using $\mathsf{Multiply}_{\mathsf{PKE}}$ or $\mathsf{Encrypt}_{\mathsf{PKE}}$;
3. determine whether the VRS signature was issued by the signer or by the sanitizer; or
4. decide whether the values $(x_j, y_j)_{j\in[\![n]\!]}$ were sampled uniformly at random or obtained by multiplying previously chosen values by random elements $r, s \leftarrow\!\!\$\, \mathbb{Z}_q^*$.

From the perfect adaptability of EQS, the unlinkability of PKE, the anonymity of VRS, and the fact that the distribution of $(x_j, y_j)_{j\in[\![n]\!]}$ is identical to that of $(s \cdot x_j,\, s \cdot r \cdot y_j)_{j\in[\![n]\!]}$. None of these advantages is achievable.

Unlinkability. Regarding unlinkability, we have to first distinguish between the notions of weak unlinkability where the adversary is only allowed to query $\mathcal{O}^b_{\mathsf{LoRSanit}}$ on honestly generated signatures and unlinkability where this restriction is removed. In order to show that the construction is unlinkable we start by proving that the adversary cannot tamper with the signature as all public information are signed using VRS which is unforgeable. This eliminates the need for the restriction on $\mathcal{O}^b_{\mathsf{LoRSanit}}$ that leads to weak unlinkability. Then, we proceed to show that the adversary cannot predict the source of a sanitized signature because all the elements in the sanitized signatures are decorrelated from the source. This is done using the following sequence of hybrid experiments: In the first hybrid, we keep history of all ciphertexts, and we use the history to do decryption instead of using $\mathsf{Decrypt}_{\mathsf{PKE}}$. In the second hybrid, we stop using $\mathsf{Multiply}_{\mathsf{PKE}}$ in the $\mathsf{Sanitize}$ algorithm and we replace it with direct encryption of the decrypted ciphertext multiplied by the scalar s. In the third hybrid, we

stop using $\mathsf{ChgRep}_{\mathsf{EQS}}$ in the Sanitize algorithm and we use $\mathsf{Sign}_{\mathsf{EQS}}$ directly. In the fourth hybrid, in the Sanitize algorithm, we replace the BLS-like verification keys $(X'_j)_{j\in[\![n]\!]}$ with fresh random values picked from G_1^n and we propagate the change to $(Y'_j)_{j\in[\![n]\!]}$ to maintain the correctness of the inner signatures. In the final and fifth hybrid, in the Sanitize algorithm, we replace the BLS-like secret keys $(y_j)_{j\in[\![n]\!]}$ with fresh random values picked from $\mathbb{Z}_q^n$ and propagate the changes to the signatures and $(Y'_j)_{j\in[\![n]\!]}$ to maintain correctness.

Then, we proceed to show that an adversary cannot distinguish any two consecutive hybrids respectively by the correctness of PKE, the unlinkability of PKE, the perfect-adaptation of EQS, and by showing that the distributions of the changed values in the fourth and fifth hybrids are indistinguishable.

Invisibility. If an adversary succeeds in breaking invisibility, this means that he was able to break the IND-CPA security of PKE by distinguishing if for some $i \in [\![k]\!], j \in [\![n]\!]$ where $a_{0,i,j} \neq a_{1,i,j}$, $\mathbf{SKZ}_{i,j}$ was generated as the encryption of the signing key or a 0.

Sanitizer Anonymity. To identify the sanitizer, the adversary can extract the information from the VRS signature which is anonymous, or in case that a message block is admissible to only one sanitizer, he can identify him by breaking the secrecy of the admissibility matrix which relies on the IND-CPA security of PKE. □

6 Full-Sanitization-Verifiable k-SAN

In the FSV-k-SAN construction, defined in Figure 4, the signer holds a pair of SIG keys, while each sanitizer possesses a key pair for VRS and a separate one for PKE. The construction uses a Chameleon hash function CHash which is common in sanitizable signatures but instead of having a permanent CHash trapdoor that is held by the sanitizer, we generate fresh keys for each message block and share the trapdoors with authorized sanitizers securely using PKE. This can be applied to existing Chameleon hash-based constructions to extend them to the k-SAN context (*e.g.,* [13]).

In the Sign algorithm, each message block is hashed using a chameleon hash function as $(j\|m_j)$ using freshly generated keys, and the resulting hash, randomness, and public key are stored in a vector $\mathbf{CH}$. A matrix of ciphertexts $\mathbf{SKCH}$ is then constructed, where $\mathbf{SKCH}_{i,j}$ is the encryption of the trapdoor for block j under the public key of sanitizer i if it is admissible for him, and an encryption of 0 otherwise. A public admissibility vector $\mathbf{PA}$ is also generated to indicate which blocks are admissible, without revealing the corresponding sanitizers. The tuple $((\mathbf{CH}_j.h, \mathbf{CH}_j.\mathsf{pk}_{\mathsf{CH}})_{j\in[\![n]\!]}, \mathbf{SKCH}, \mathbf{PA}, \mathsf{pk}_{\mathsf{S}}, \mathbf{PKZ}, n)$ is then signed using SIG, yielding the signature s. Additionally, the signer creates a proof for each message block by signing $(j\|m_j\|s)$ and storing the resulting signatures in a vector ρ. Including s in these signatures prevents reuse of prior signatures over the

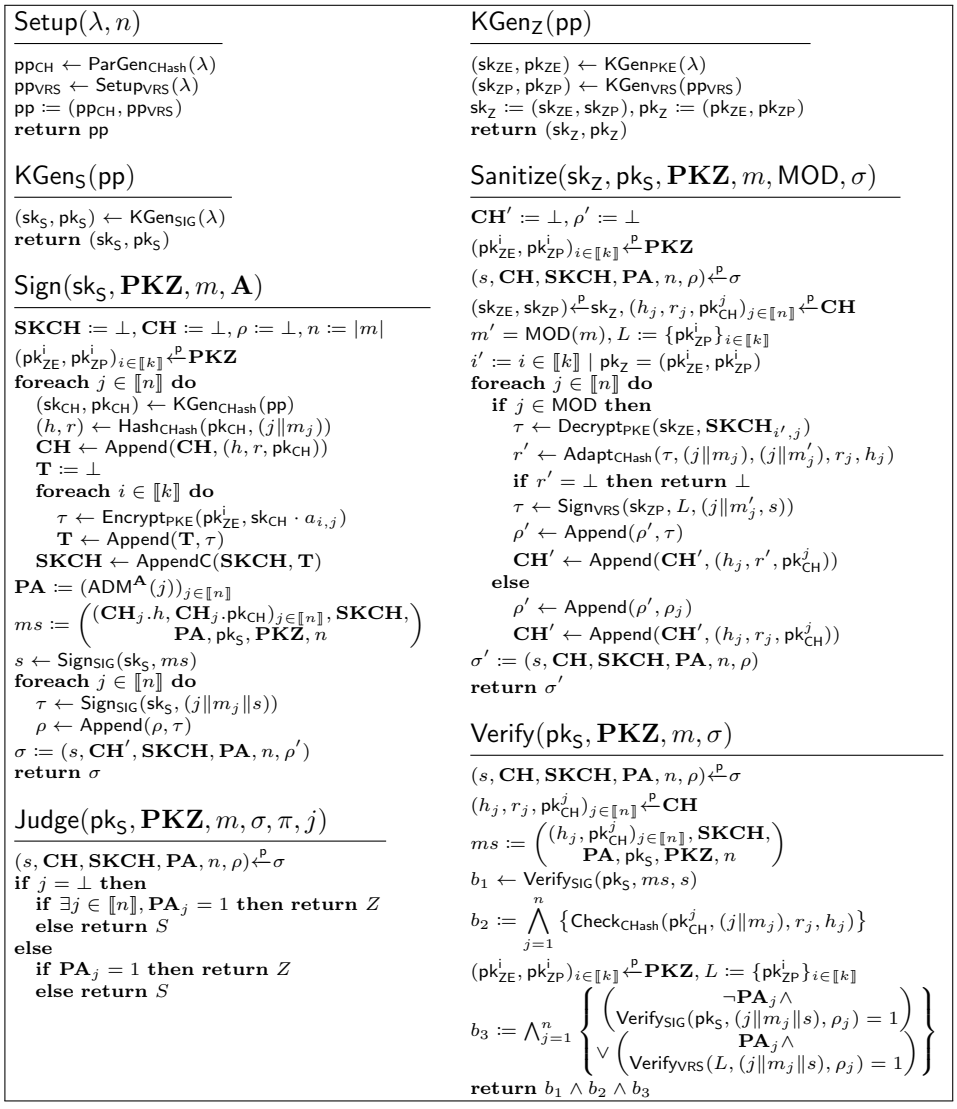

Fig. 4: Algorithms of the FSV-k-SAN construction.

same message block in a different k-SAN context. The final signature consists of the tuple $(s, \mathbf{CH}, \mathbf{SKCH}, \mathbf{PA}, n, \rho)$.

Signatures coming from Sign can be sanitized based on the defined admissibility policy using the Sanitize algorithm. The sanitizer decrypts their corresponding entries in **SKCH** to retrieve the trapdoors, which are then used to adapt the relevant message blocks and their associated randomness via the $\mathsf{Adapt}_{\mathsf{CHash}}$ algorithm. Each modified block is then signed using VRS to prove that the sanitization was performed by a member of the ring formed by all sanitizers' public keys. The VRS signature is computed over the tuple $(j \| m_j \| s)$. The algorithm finally returns the updated signature, including the modified chameleon hash randomness and the sanitization proofs collected in ρ.

The Verify algorithm ensures the integrity of a signature by first checking the main signature s, then verifying each chameleon hash using $\mathsf{Check}_{\mathsf{CHash}}$. It further verifies that every admissible block (as indicated by **PA**) is accompanied by a valid VRS signature, while inadmissible blocks must carry a valid SIG signature.

In this construction, the Prove algorithm is not required due to full-sanitization verifiability which allows the Judge algorithm to function without explicit proofs. Consequently, the oracle $\mathcal{O}_{\mathsf{Prove}}$ always returns $\perp$. We consider that the Judge algorithm is called on valid signatures only, which implicitly confirms that each message block has a valid proof—either of signing or sanitization. If $j = \perp$, the algorithm returns Z if at least one block is admissible, and S otherwise. If $j \neq \perp$, it returns Z if $\mathbf{PA}_j = 1$, and S otherwise.

While VRS could be replaced by standard ring signatures in this construction, we retain VRS to support future extensions where identifying the sanitizer of a block is required.

We also provide high-level flowcharts that show how the Sign and Sanitize algorithms work in IUT-k-SAN and FSV-k-SAN in Appendix D to better highlight the differences.

Theorem 4. FSV-k-SAN *verifies the security properties of correctness, unforgeability, immutability, accountability, privacy, full-sanitization verifiability, and sanitizer anonymity, if the underlying building blocks* CHash*,* SIG*,* PKE*, and* VRS *are secure.*

Proof. The security proofs can be found in Appendix E. Here, we will explain the general idea of the proofs. As with the previous scheme, the correctness follows directly from that of the underlying building blocks and unforgeability is implied by accountability.

Immutability. If an adversary succeeds in breaking immutability, this means that he was able to change one or more inadmissible message blocks. We use this to construct adversaries against the security properties of the underlying building blocks thus eliminating all possible axes of attack if the building blocks are secure. Concretely breaking immutability implies either forging a SIG signature

where the signed hash values were changed (*i.e.,* breaking the EUF-CMA security of SIG), breaking the encryption of the chameleon hash trapdoors (*i.e.,* breaking the IND-CPA security of PKE), finding a collision for the chameleon hashes without access to the trapdoor (*i.e.,* breaking the collision-resistance of CHash), or finding a different CHash randomness for the same message thus breaking its uniqueness.

Accountability. We do the proof by doing a reduction from the accountability games to the unforgeability of VRS and SIG. If an adversary succeeds in the signer accountability game, this means that he was able to provide a VRS signature under the ring of sanitizers. On the other hand, winning in the signer accountability game means that the adversary was able to produce a SIG forgery as a signature proof for some message block.

Privacy. The signature generated in the Sign algorithm, contains two elements related to the message, the SIG signature proof and the chameleon hash value. The SIG signature is overwritten by the VRS signature which means it does not leak information about the original message. The hash value also does not leak information because of the indistinguishability of CHash. We show this by doing a reduction from the privacy game to CHash's indistinguishability.

Full-Sanitization Verifiability. As in the Verify algorithm we check for a VRS signature of every admissible block, an adversary that has a non-negligible probability in winning the game can be used to construct an adversary against the unforgeability of VRS.

Sanitizer Anonymity. To reveal the identity of the sanitizer, the adversary needs to extract the information from the VRS signature which is anonymous. He has another option in case that the message block is admissible to only one sanitizer. In this case, if he can break the IND-CPA security of PKE, he can know to which of the k sanitizers the trapdoor was encrypted instead of a 0. □

7 Implementation and Performance Analysis

For IUT-k-SAN, we instantiate it with the VRS scheme of Bultel and Lafourcade [10], Mercurial signatures [17] for EQS, and the Paillier cryptosystem [27] for PKE. To be able to compare fairly our two implementations, we use for FSV-k-SAN the same PKE and VRS as IUT-k-SAN, Schnorr signatures [31], and the discrete log chameleon hash construction from [23]. The theoretical efficiency of both constructions can be found in Appendix F.

We implemented both constructions in Rust and tested their performance on a Linux server with an Intel i5-11500 processor and 32GB of RAM, and a Google Pixel 6a smartphone. Regarding the building blocks, we implemented the BLS-like signatures, CHash and VRS using the num_bigint, glass_pumpkin, and ark-bls12-381 Rust crates, and used the existing crates kzen-paillier, k256, and delegatable_credentials for PKE, SIG, and EQS respectively. Figure 5 shows the performance for different values of n (number of message blocks) in both

constructions while fixing the number of sanitizers k to 5. Although we have only 3 sanitizers in the multi-party contract and medical document use cases, we use a slightly bigger value to allow for cases where more sanitizers are needed. For example, the rental contract can involve additional parties like an insurance company and a second guarantor. The execution time of the Sanitize algorithm depends on the number of modified blocks which was fixed to 1 in each call to the algorithm. As Verify in FSV-k-SAN depends on the number of admissible blocks, we fixed it to 5. Regarding the security parameters, we set $\lambda = 2048$ for CHash and VRS, and 2056 for PKE. We use the secp256k1 curve for SIG and BLS12-381 for EQS and the BLS-like signatures. We use a slightly bigger λ for PKE to make sure that the modulus is big enough to encrypt the CHash secret keys. The results were obtained by executing each algorithm 200 times and calculating the average execution time.

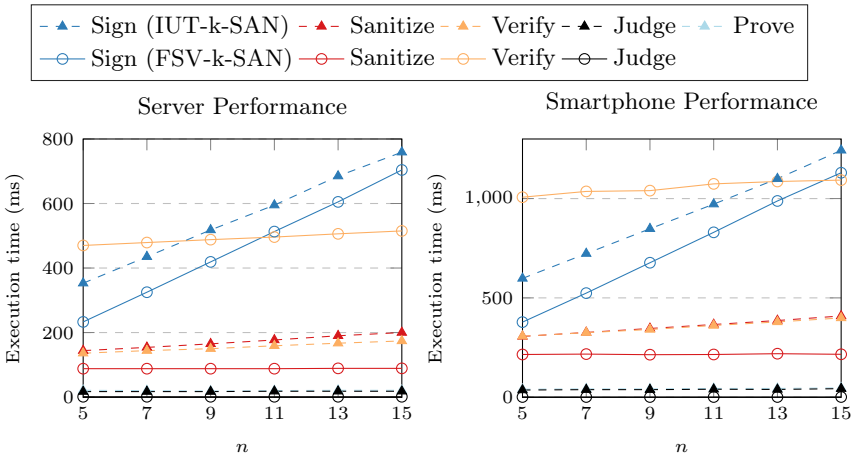

Fig. 5: Performance of the implementation of both constructions. We fix $k = 5$ and test with variable n. The first figure shows the performance on a server, while the second figure shows the performance on a smartphone.

On the server, the Sign algorithm took ∼704 ms on $n = 15$ while IUT-k-SAN took ∼759 ms. Verify in FSV-k-SAN took ∼515 ms. Other algorithms in both constructions took less than ∼205 ms. Regarding the smartphone, signing a message with $n = 15$ took ∼1130 ms and ∼1243 ms in FSV-k-SAN and IUT-k-SAN respectively. Verify took ∼1093 ms in FSV-k-SAN. Other algorithms took less than ∼410 ms in both constructions. This degradation of performance is expected as we pass to a less powerful device. The performance is acceptable for many applications that require the advanced features and security properties that we propose.

We notice that compared to the theoretical analysis, the real execution time of IUT-k-SAN is for the most part not much different from FSV-k-SAN. This is due to using a large λ for PKE, CHash, and VRS which makes the contribution of the other building blocks negligible as the fields and groups in the used elliptic curves are much smaller. Thus, we did another test using a smaller $\lambda = 512$ for CHash and VRS and 520 for PKE to have a more accurate comparison between theoretical and real performance even if these values are insecure. We show the results of this test in Appendix F. During this test we found that the implementation of PKE in the kzen-paillier crate is faster than directly doing exponentiation in the num_bigint crate. Moreover, the delegatable_credentials crate uses the multi Miller loop optimization [32] to do the multiple pairings in $\mathsf{Verify}_{\mathsf{EQS}}$ which leads to a significant performance gain on the server. However, this optimization did not perform well on the smartphone. We tested another crate called mercurial-signature for EQS which does not use the multi Miller loop and found that it performs better than the delegatable_credentials crate on the smartphone but worse on the server. We maintained the use of the delegatable_credentials crate as the gain of performance on the server side was more significant than the loss on the smartphone side. Precisely, the verification of a signature on a 15 elements message in the delegatable_credentials crate took $\sim$4 ms on the server and $\sim$43 ms on the smartphone, while the mercurial-signature crate took $\sim$20 ms on the server and $\sim$30 ms on the smartphone.

We also compare the efficiency of our constructions with the existing multi-sanitizer schemes [1, 7, 14, 15, 25, 30]. We differ the comparison to Appendix G as it is hard to give an accurate conclusion of which schemes are more efficient given that our constructions and the existing ones support different security properties and the existing constructions have a unique admissibility policy for all sanitizers.

8 Conclusion

We introduced a multi-sanitizer sanitizable signature scheme which allows different sanitizers to have different admissibility policies. We defined its security model, introduced a new property called full-sanitization verifiability, and proved that it breaks proof-restricted transparency and invisibility. We designed two generic constructions IUT-k-SAN and FSV-k-SAN with different security properties adapted to different use cases and proved their security. Finally, we implemented the constructions using Rust and shown that they have acceptable performance on a server and a smartphone.

Our implementation uses pre-quantum primitives, but FSV-k-SAN could employ post-quantum ones such as the Chameleon hash and VRS from [16], Kyber [4], and Dilithium [20]. Since the Chameleon hash of [16] lacks uniqueness, we can sign the randomness using SIG and VRS in the block-wise signing/sanitization proofs to compensate for this. Post-quantum security is not possible for IUT-k-SAN because EQS has no post-quantum alternative yet.

Acknowledgments. We would like to thank Anthony Graignic for his assistance with the implementation and the anonymous reviewers for their useful feedback. This work has been partially supported by the French National Research Agency through the PRIvacy-preserving tools for VAlidation and Security of Queries (PRIVASIQ) project (ANR-23-CE39-0008).

References

1. I. Afia and R. AlTawy. Unlinkable policy-based sanitizable signatures. In M. Rosulek, editor, *Topics in Cryptology – CT-RSA 2023*, volume 13871 of *Lecture Notes in Computer Science*, pages 191–221, San Francisco, CA, USA, Apr. 24–27, 2023. Springer, Cham, Switzerland.
2. G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In S. De Capitani di Vimercati, P. F. Syverson, and D. Gollmann, editors, *ESORICS 2005: 10th European Symposium on Research in Computer Security*, volume 3679 of *Lecture Notes in Computer Science*, pages 159–177, Milan, Italy, Sept. 12–14, 2005. Springer Berlin Heidelberg, Germany.
3. N. Attrapadung, J. Herranz, F. Laguillaumie, B. Libert, E. De Panafieu, and C. Ràfols. Attribute-based encryption schemes with constant-size ciphertexts. *Theoretical computer science*, 422:15–38, 2012.
4. J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. Crystals-kyber: a cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018.
5. A. Bossuat and X. Bultel. Unlinkable and invisible γ-sanitizable signatures. In K. Sako and N. O. Tippenhauer, editors, *ACNS 2021: 19th International Conference on Applied Cryptography and Network Security, Part I*, volume 12726 of *Lecture Notes in Computer Science*, pages 251–283, Kamakura, Japan, June 21–24, 2021. Springer, Cham, Switzerland.
6. C. Brzuska, M. Fischlin, T. Freudenreich, A. Lehmann, M. Page, J. Schelbert, D. Schröder, and F. Volk. Security of sanitizable signatures revisited. In S. Jarecki and G. Tsudik, editors, *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 317–336, Irvine, CA, USA, Mar. 18–20, 2009. Springer Berlin Heidelberg, Germany.
7. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Sanitizable signatures: How to partially delegate control for authenticated data. 2009.
8. C. Brzuska, M. Fischlin, A. Lehmann, and D. Schröder. Unlinkability of sanitizable signatures. In P. Q. Nguyen and D. Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 444–461, Paris, France, May 26–28, 2010. Springer Berlin Heidelberg, Germany.
9. C. Brzuska, H. C. Pöhls, and K. Samelin. Efficient and perfectly unlinkable sanitizable signatures without group signatures. In *Public Key Infrastructures, Services and Applications: 10th European Workshop*, 2014.
10. X. Bultel and P. Lafourcade. Unlinkable and strongly accountable sanitizable signatures from verifiable ring signatures. In S. Capkun and S. S. M. Chow, editors, *CANS 17: 16th International Conference on Cryptology and Network Security*, volume 11261 of *Lecture Notes in Computer Science*, pages 203–226, Hong Kong, China, Nov. 30 – Dec. 2, 2017. Springer, Cham, Switzerland.

11. X. Bultel, P. Lafourcade, R. W. F. Lai, G. Malavolta, D. Schröder, and S. A. K. Thyagarajan. Efficient invisible and unlinkable sanitizable signatures. In D. Lin and K. Sako, editors, *PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 11442 of *Lecture Notes in Computer Science*, pages 159–189, Beijing, China, Apr. 14–17, 2019. Springer, Cham, Switzerland.
12. X. Bultel and C. Olivier-Anclin. Taming delegations in anonymous signatures: k-times anonymity for proxy and sanitizable signature. In M. Kohlweiss, R. Di Pietro, and A. R. Beresford, editors, *CANS 2024: 23rd International Conference on Cryptology and Network Security, Part I*, volume 14905 of *Lecture Notes in Computer Science*, pages 165–186, Cambridge, UK, Sept. 24–27, 2024. Springer, Singapore, Singapore.
13. J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig. Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures. In S. Fehr, editor, *PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 10175 of *Lecture Notes in Computer Science*, pages 152–182, Amsterdam, The Netherlands, Mar. 28–31, 2017. Springer Berlin Heidelberg, Germany.
14. S. Canard, A. Jambert, and R. Lescuyer. Sanitizable signatures with several signers and sanitizers. In A. Mitrokotsa and S. Vaudenay, editors, *AFRICACRYPT 12: 5th International Conference on Cryptology in Africa*, volume 7374 of *Lecture Notes in Computer Science*, pages 35–52, Ifrance, Morocco, July 10–12, 2012. Springer Berlin Heidelberg, Germany.
15. S. Canard, F. Laguillaumie, and M. Milhau. Trapdoor sanitizable signatures and their application to content protection. In *Applied Cryptography and Network Security: 6th International Conference, ACNS 2008, New York, NY, USA, June 3-6, 2008. Proceedings 6*, pages 258–276. Springer, 2008.
16. S. Clermont, S. Düzlü, C. Janson, L. Porzenheim, and P. Struck. Lattice-based sanitizable signature schemes: Chameleon hash functions and more. In *International Conference on Post-Quantum Cryptography*, pages 278–311. Springer, 2025.
17. E. C. Crites and A. Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In M. Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, volume 11405 of *Lecture Notes in Computer Science*, pages 535–555, San Francisco, CA, USA, Mar. 4–8, 2019. Springer, Cham, Switzerland.
18. D. Derler, K. Samelin, D. Slamanig, and C. Striecks. Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. In *ISOC Network and Distributed System Security Symposium – NDSS 2019*, San Diego, CA, USA, Feb. 24–27, 2019. The Internet Society.
19. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
20. L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.
21. N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schröder, and M. Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In C.-M. Cheng, K.-M. Chung, G. Persiano, and B.-Y. Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 9614 of *Lecture Notes in Computer Science*,

pages 301–330, Taipei, Taiwan, Mar. 6–9, 2016. Springer Berlin Heidelberg, Germany.
22. C. Hanser and D. Slamanig. Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 491–511, Kaoshiung, Taiwan, R.O.C., Dec. 7–11, 2014. Springer Berlin Heidelberg, Germany.
23. H. Krawczyk and T. Rabin. Chameleon hashing and signatures. *Internet*, 1997.
24. S. Krenn, K. Samelin, and D. Sommer. Stronger security for sanitizable signatures. In *International Workshop on Data Privacy Management*, pages 100–117. Springer, 2015.
25. J. Lai, X. Ding, and Y. Wu. Accountable trapdoor sanitizable signatures. In *Information Security Practice and Experience: 9th International Conference, ISPEC 2013, Lanzhou, China, May 12-14, 2013. Proceedings 9*, pages 117–131. Springer, 2013.
26. J. Lv and X. Wang. Verifiable ring signature. In *Proceedings 9th International Conference on Distributed Media Systems (DMS 2003, Miami FL, USA, September 24-26, 2003; 3rd International Workshop on Cryptology and Network Security (CANS 2003))*, pages 663–665, 2003.
27. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer Berlin Heidelberg, Germany.
28. M. Prabhakaran and M. Rosulek. Homomorphic encryption with cca security. In *International Colloquium on Automata, Languages, and Programming*, pages 667–678. Springer, 2008.
29. R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565, Gold Coast, Australia, Dec. 9–13, 2001. Springer Berlin Heidelberg, Germany.
30. K. Samelin and D. Slamanig. Policy-based sanitizable signatures. In S. Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, volume 12006 of *Lecture Notes in Computer Science*, pages 538–563, San Francisco, CA, USA, Feb. 24–28, 2020. Springer, Cham, Switzerland.
31. C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, Jan. 1991.
32. M. Scott. Pairing implementation revisited. Cryptology ePrint Archive, Report 2019/077, 2019.
33. V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer Berlin Heidelberg, Germany.

A Building Blocks Algorithms

Definition 12 (CHash **[13, 23]).** *A* chameleon-hash CHash *consists of five algorithms which are defined as follows:*

$\mathsf{pp}_{\mathsf{CH}} \leftarrow \mathsf{ParGen}_{\mathsf{CHash}}(\lambda)$: *On input the security parameter λ, the algorithm outputs the public parameter* $\mathsf{pp}_{\mathsf{CH}}$.

$(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}}) \leftarrow \mathsf{KGen}_{\mathsf{CHash}}(\mathsf{pp}_{\mathsf{CH}})$: *On input the public parameter* $\mathsf{pp}_{\mathsf{CH}}$*, the algorithm outputs a tuple* $(\mathsf{sk}_{\mathsf{CH}}, \mathsf{pk}_{\mathsf{CH}})$ *containing the secret key (trapdoor)* $\mathsf{sk}_{\mathsf{CH}}$ *and the public key* $\mathsf{pk}_{\mathsf{CH}}$*.*

$(h, r) \leftarrow \mathsf{Hash}_{\mathsf{CHash}}(\mathsf{pk}_{\mathsf{CH}}, m)$: *On input a public key* $\mathsf{pk}_{\mathsf{CH}}$ *and a message* m*, the algorithm outputs a tuple* (h, r) *containing a hash* h *and a randomness* r*.*

$b \leftarrow \mathsf{Check}_{\mathsf{CHash}}(\mathsf{pk}_{\mathsf{CH}}, m, r, h)$: *On input a public key* $\mathsf{pk}_{\mathsf{CH}}$*, a message* m*, a randomness* r*, and a hash* h*, the algorithm outputs a bit* b *indicating if the given hash is valid.*

$r' \leftarrow \mathsf{Adapt}_{\mathsf{CHash}}(\mathsf{sk}_{\mathsf{CH}}, m, m', r, h)$: *On input a secret key* $\mathsf{sk}_{\mathsf{CH}}$*, a message* m*, a new message* m'*, a randomness* r*, and a hash* h*, the algorithm outputs a new randomness* r'*, such that* $\mathsf{Check}_{\mathsf{CHash}}(\mathsf{pk}_{\mathsf{CH}}, m, r, h) = \mathsf{Check}_{\mathsf{CHash}}(\mathsf{pk}_{\mathsf{CH}}, m', r', h) = 1$*.*

Definition 13 (SIG**).** *A* signature scheme SIG *consists of three algorithms which are defined as follows:*

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{SIG}}(\lambda)$: *On input the security parameter* λ*, the algorithm outputs a tuple* $(\mathsf{sk}, \mathsf{pk})$ *containing* sk *the secret key and* pk *the public key.*

$\sigma \leftarrow \mathsf{Sign}_{\mathsf{SIG}}(\mathsf{sk}, m)$: *On input a secret key* sk *and a message* m*, the algorithm outputs a signature* σ*.*

$b \leftarrow \mathsf{Verify}_{\mathsf{SIG}}(\mathsf{pk}, m, \sigma)$: *On input a public key* pk*, a message* m *and a signature* σ*, the algorithm outputs a bit* b *indicating if the signature is valid.*

Definition 14 (PKE**).** *A* public-key asymmetric encryption scheme PKE *consists of the algorithms* $(\mathsf{KGen}_{\mathsf{PKE}}, \mathsf{Encrypt}_{\mathsf{PKE}}, \mathsf{Decrypt}_{\mathsf{PKE}})$*. If* PKE *is homomorphic for scalar multiplication, then a fourth algorithm* $\mathsf{Multiply}_{\mathsf{PKE}}$ *is added. The algorithms are defined as follows:*

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{PKE}}(\lambda)$: *On input the security parameter* λ*, the algorithm outputs a tuple* $(\mathsf{sk}, \mathsf{pk})$ *containing* sk *the secret key and* pk *the public key.*

$c \leftarrow \mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}, m)$: *On input a public key* pk *and a plaintext* m*, the algorithm outputs a ciphertext* c*.*

$m \leftarrow \mathsf{Decrypt}_{\mathsf{PKE}}(\mathsf{sk}, c)$: *On input a secret key* sk *and a ciphertext* c*, the algorithm outputs the decrypted plaintext* m*.*

$c' \leftarrow \mathsf{Multiply}_{\mathsf{PKE}}(\mathsf{pk}, c, s)$: *On input a public key* pk*, a ciphertext* c *of a plaintext* m *and a scalar* s*, the algorithm outputs a new ciphertext* c' *that decrypts to* $s \cdot m$*.*

Definition 15 (VRS **[10, 11]).** *A* verifiable ring signature scheme VRS *consists of six algorithms which are defined as follows:*

$\mathsf{pp}_{\mathsf{VRS}} \leftarrow \mathsf{Setup}_{\mathsf{VRS}}(\lambda)$: *On input the security parameter* λ*, the algorithm outputs the public parameter* $\mathsf{pp}_{\mathsf{VRS}}$*.*

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{VRS}}(\mathsf{pp}_{\mathsf{VRS}})$: *On input the public parameter* $\mathsf{pp}_{\mathsf{VRS}}$*, the algorithm outputs a tuple* $(\mathsf{sk}, \mathsf{pk})$ *containing* sk *the secret key and* pk *the public key.*

$\sigma \leftarrow \mathsf{Sign}_{\mathsf{VRS}}(\mathsf{sk}, L, m)$: *On input a secret key* sk*, a ring* L *which is a set of public keys, and a message* m*, the algorithm outputs the signature* σ*.*

$b \leftarrow \mathsf{Verify}_{\mathsf{VRS}}(L, m, \sigma)$: *On input a ring L, a message m, and a signature σ, the algorithm outputs a bit b indicating if the signature is valid.*

$\pi \leftarrow \mathsf{Prove}_{\mathsf{VRS}}(L, m, \sigma, \mathsf{pk}, \mathsf{sk})$: *On input a ring L, a message m, a signature σ, a public key pk, and a secret key sk, the algorithm outputs a proof π.*

$b \leftarrow \mathsf{Judge}_{\mathsf{VRS}}(L, m, \sigma, \mathsf{pk}, \pi)$: *On input a ring L, a message m, a signature σ, a public key pk, and a proof π, the algorithm outputs a bit b indicating if the signature was or was not generated by the signer pk.*

Definition 16 (Bilinear Map [22]). *Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order q where $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive and $\mathbb{G}_T$ is multiplicative. Let g_1 and g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We call $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ a bilinear map or pairing if it is efficiently computable and the following conditions hold:*

Bilinearity: $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab} = e(g_1^b, g_2^a), \forall a, b \in \mathbb{Z}_q$,
Non-degeneracy: $e(g_1, g_2) \neq 1_{\mathbb{G}^T}$, i.e.,, $e(g_1, g_2)$ *generates* $\mathbb{G}^T$.

Definition 17 (EQS [11, 22]). *An equivalence class signature (EQS) scheme is a tuple of six algorithms defined as follows:*

$\mathcal{BG} \leftarrow \mathsf{BGGen}(\lambda)$: *On input the security parameter λ, the algorithm outputs $\mathcal{BG} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, g_T, e, q)$ the description of a multiplicative bilinear group of prime order q.*

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{EQS}}(\mathcal{BG}, n)$: *On input the group $\mathcal{BG}$ and the message length n, the algorithm outputs a tuple $(\mathsf{sk}, \mathsf{pk})$ containing sk the secret key and pk the public key.*

$\sigma \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}, \bar{M})$: *On input a secret key sk and a message $\bar{M} \in \mathbb{G}_1^n$, the algorithm outputs a signature σ on the equivalence class $[\bar{M}]_{\mathcal{R}}$.*

$\sigma' \leftarrow \mathsf{ChgRep}_{\mathsf{EQS}}(\mathsf{pk}, \bar{M}, \sigma, \rho)$: *On input a public key pk, a message $\bar{M} \in \mathbb{G}_1^n$, a signature σ on the equivalence class $[\bar{M}]_{\mathcal{R}}$, and a scalar ρ, the algorithm outputs a new signature σ' on the (same) equivalence class $[\bar{M}^\rho]_{\mathcal{R}} = [\bar{M}]_{\mathcal{R}}$.*

$b \leftarrow \mathsf{Verify}_{\mathsf{EQS}}(\mathsf{pk}, \bar{M}, \sigma)$: *On input a public key pk, a message $\bar{M} \in \mathbb{G}_1^n$, and a signature σ, the algorithm outputs a bit b indicating if the signature is valid.*

$b \leftarrow \mathsf{VerifyKey}_{\mathsf{EQS}}(\mathsf{sk}, \mathsf{pk})$: *On input a secret key sk and a public key pk, the algorithm outputs a bit b indicating if the keys are consistent.*

B Security Experiments for The Building Blocks

All the security experiments and oracles mentioned in the definitions in this section are defined in Figure 6.

B.1 Properties of Chameleon Hashes

Collision-Resistence. Collision-Resistence requires that it is hard for an adversary to find a collision without knowing the trapdoor.

Definition 18 (Collision-Resistence of CHash **[13]).** *A chameleon hash* CHash *is collision-resistent if for all* PPT *adversaries* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Col-Res}}_{\mathrm{CHash},\mathcal{A}}(\lambda)=1\right] \leq \mathsf{negl}(\lambda)$.

Uniqueness. Uniqueness requires that it is hard for an adversary to find two different valid randomness values for the same message and hash value.

Definition 19 (Uniqueness of CHash **[13]).** *A chameleon hash* CHash *is unique if for all* PPT *adversaries* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Uniq}}_{\mathrm{CHash},\mathcal{A}}(\lambda)=1\right] \leq \mathsf{negl}(\lambda)$.

Indistinguishability. Indistinguishability requires that the randomness does not reveal if it was obtained from $\mathsf{Hash}_{\mathsf{CHash}}$ or $\mathsf{Adapt}_{\mathsf{CHash}}$.

Definition 20 (Indistinguishability of CHash **[13]).** *A chameleon hash* CHash *is indistinguishable if for all* PPT *adversaries* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Ind},b}_{\mathrm{CHash},\mathcal{A}}(\lambda)=1\right] \leq \mathsf{negl}(\lambda)$.

B.2 Properties of Signatures

Definition 21 (EUF-CMA security of SIG**).** *A signature scheme* SIG *is existentially unforgeable under chosen message attacks (EUF-CMA) if for all* PPT *adversaries* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{EUF-CMA}}_{\mathrm{SIG},\mathcal{A}}(\lambda)=1\right] \leq \mathsf{negl}(\lambda)$.

B.3 Properties of Public Key Encryption

We require IND-CPA security for both constructions and unlinkability for the IUT-k-SAN construction.

Definition 22 (IND-CPA Security). *A* PKE *scheme has indistinguishability against chosen-plaintext attacks (IND-CPA) if for all* PPT *adversaries* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{IND-CPA},b}_{\mathrm{PKE},\mathcal{A}}(\lambda)=1\right] \leq \mathsf{negl}(\lambda)$.

Definition 23 (Unlinkability of PKE [28]). *A* PKE *scheme is unlinkable if for all* PPT *adversaries* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Unlink},b}_{\mathrm{PKE},\mathcal{A}}(\lambda)=1\right] \leq \mathsf{negl}(\lambda)$.

B.4 Properties of Verifiable Ring Signatures

In the following definitions, $\mathcal{Q}$ is the set of message-signature pairs associated with the queries to the oracle $\mathcal{O}_{\mathsf{Sign}_{\mathsf{VRS}}}$.

Unforgeability. A VRS is unforgeable when no adversary that has access to the sign and proof oracles is able to forge a fresh message-signature pair (m, σ), such that the corresponding ring contains public keys of honest users only.

Definition 24 (Unforgeability of VRS [10, 11]). *A VRS scheme* VRS *is unforgeable if for all* $n \in \mathsf{poly}(\lambda)$ *and for all* PPT *adversary* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Unforg},n}_{\mathrm{VRS},\mathcal{B}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$.

Anonymity. A VRS is anonymous when no adversary can link a signature to the public key of its signer. The adversary has access to the $\mathcal{O}_{\mathsf{Sign}_{\mathsf{VRS}}}$, $\mathcal{O}_{\mathsf{Prove}_{\mathsf{VRS}}}$, and $\mathcal{O}^b_{\mathsf{LoRSign}_{\mathsf{VRS}}}$ oracles.

Definition 25 (Anonymity of VRS [10, 11]). *A VRS scheme* VRS *is anonymous if for all* PPT $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Anon},n,b}_{\mathrm{VRS},\mathcal{B}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$.

Accountability. A VRS is accountable when no adversary that has access to the signature oracle and the proof oracle is able to forge a fresh message-signature pair (m, σ) together with a proof that it is not the signer of σ, such that the corresponding ring contains at most one public key of a non-honest user.

Definition 26 (Accountability of VRS [10, 11]). *A VRS scheme* VRS *is accountable if for all* $n \in \mathsf{poly}(\lambda)$ *and for all* PPT *adversary* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Acc},n}_{\mathrm{VRS},\mathcal{B}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$.

Non-Seizability. A VRS is non-seizable when no adversary that has access to the signature and the proof oracle is able to forge a fresh message-signature pair (m, σ), such that the proof algorithm ran by the honest user returns a proof that σ was computed by the honest user.

Definition 27 (Non-Seizability of VRS [10, 11]). *A VRS scheme* VRS *is non-seizable if for all* PPT *adversary* $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Non-Seiz}}_{\mathrm{VRS},\mathcal{B}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$.

B.5 Properties of Equivalence Class Signatures

We recall the definitions of Class-Hiding Groups and the security properties of EQS from [11]. Class-hiding was introduced by Hanser and Slamanig [22] as a property of equivalence class signatures.

Class-Hiding Groups. Let $\mathcal{BG} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, g_T, e, q) \leftarrow \mathsf{BGGen}(\lambda)$ be the description of a multiplicative bilinear group of prime order q generated by some efficient PPT algorithm $\mathsf{BGGen}(\lambda)$. Let $\bar{X} = (X_1, \ldots, X_n) \in \mathbb{G}_1^n$ and $\rho \in \mathbb{Z}_q$. We write $\bar{X} := (X_1, \ldots, X_n)^\rho := (X_1^\rho, \ldots, X_n^\rho)$. We then define the equivalence relation

$$\mathcal{R} := \{(\bar{M}, \bar{N}) : \exists n > 1,\ \rho \in \mathbb{Z}_q^*\ s.t.\ (\bar{M}, \bar{N}) \in \mathbb{G}_1^n \times \mathbb{G}_1^n \wedge \bar{N} = \bar{M}^\rho\}.$$

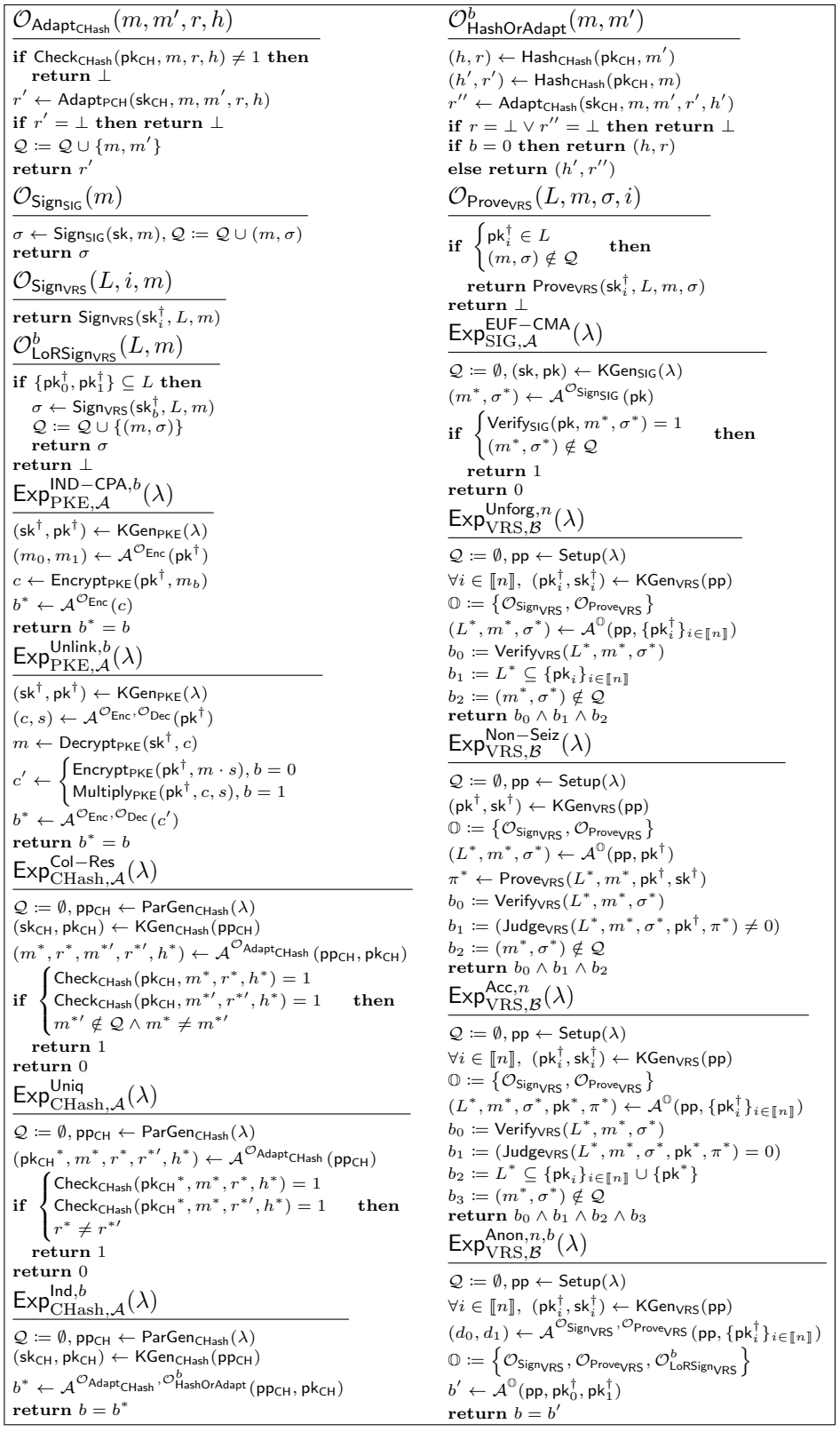

Fig. 6: Security experiments and oracles for the building blocks.

For a vector $\bar{M} \in \mathbb{G}_1^n$ for some $n > 1$, its equivalence class is defined by $[\bar{M}]_{\mathcal{R}} := \{\bar{N} \in \mathbb{G}_1^n : (\bar{M}, \bar{N}) \in \mathcal{R}\}$. A relation $\mathcal{R}$ is class-hiding if it is hard to distinguish elements from the same equivalence class from randomly sampled group elements.

Definition 28 (Class-Hiding). *A relation $\mathcal{R}$ is class-hiding if for all $n > 1$ and for all* PPT *adversaries $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\mathsf{Pr}\left[b' = b : \begin{array}{l} b \leftarrow \{0,1\}; \mathcal{BG} \leftarrow \mathsf{BGGen}(\lambda); (M, M_0) \leftarrow\!\$\, (\mathbb{G}_1^n)^2; \\ M_1 \leftarrow [M]_{\mathcal{R}}; b' \leftarrow \mathcal{A}(\mathcal{BG}, M, M_b) \end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

Definition 29 (EUF-CMA). *An EQS scheme is existentially unforgeable under chosen message attacks (EUF-CMA) if for all $n > 1$, for all $n \in \mathsf{poly}(\lambda)$, and for all* PPT *adversaries $\mathcal{A}$,*

$$\mathsf{Pr}\left[\begin{array}{l} 1 = \mathsf{Verify}(\mathsf{pk}, M^*, \sigma^*) \wedge \\ \forall M \in Q : [M]_{\mathcal{R}} \neq [M^*]_{\mathcal{R}} \end{array} : \begin{array}{l} \mathcal{BG} \leftarrow \mathsf{BGGen}(\lambda); \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}_{\mathsf{EQS}}(\mathcal{BG}, 1^n); \\ (M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}, \cdot)}(\mathsf{pk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Perfect signature adaptation means signatures where the representation was changed by the $\mathsf{ChgRep}_{\mathsf{EQS}}$ algorithm are distributed like fresh signatures.

Definition 30 (Perfect Signature Adaptation). *An EQS scheme perfectly adapts signatures if for all tuples $(\mathsf{sk}, \mathsf{pk}, \bar{M}, \sigma, \rho)$ such that $\mathsf{VerifyKey}_{\mathsf{EQS}}(\mathsf{pk}, \mathsf{sk}) = 1$, $\mathsf{Verify}_{\mathsf{EQS}}(\mathsf{pk}, \bar{M}, \sigma) = 1$, $\bar{M} \in \mathbb{G}_1^n$ for some $n > 1$, and $\rho \leftarrow \mathbb{Z}_q^*$ it holds that $\mathsf{ChgRep}_{\mathsf{EQS}}(\mathsf{pk}, \bar{M}, \sigma, \rho)$ and $\mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}, \bar{M}^\rho)$ are identically distributed.*

C Proofs for The IUT-k-SAN Construction

As our construction builds on the concepts explored by Bultel *et al.* [11] and uses their BLS-like signatures, we can reuse a good portion of their proofs. We recall that in [11], Bultel *et al.* have two constructions Π_1 which does not include VRS and Π_2 with VRS. Π_1 has weak-immutability where the adversary can tamper with the sanitizer's public key, perfect strong transparency, strong invisibility, and weak unlinkability. Π_2 has immutability where the adversary cannot tamper with the sanitizer's public key, strong accountability, strong proof-restricted transparency, strong invisibility, and unlinkability. The resused proofs have been adapted to our model and according to the changes that allowed multiple sanitizers with different admissibility policies.

Bultel *et al.*'s [11] proofs use the generic group model abstraction of Shoup [33]. The following lemma is used to prove facts about generic attackers.

Lemma 2 (Schwartz-Zippel). *Let $F(X_1, \ldots, X_m)$ be a non-zero polynomial of degree $d \geq 0$ over a field $\mathbb{F}$. Then the probability that $F(x_1, \ldots, x_m) = 0$ for randomly chosen values $(x_1, \ldots, x_m)$ in $\mathbb{F}^n$ is bounded from above by $\frac{d}{|\mathbb{F}|}$.*

Proof (Correctness.). The correctness of k-SAN relies directly on the correctness of the underlying signature schemes VRS, EQS, and BLS-like. □

Before starting the immutability proof, we recall that the challenger of our immutability game generates a set of sanitizers $\mathbf{PKZ}^{\dagger}$ to which the adversary does not have access and that the adversary can generate his own sanitizers if needed.

Proof (Immutability). The first part of our proof (proof of Lemma 1) is a copy of Bultel *et al.*'s [11] proof of weak immutability for Π_1. In their proof they upgraded weak immutability to immutability by signing the sanitizer public key with the message. We did the same in our construction. We modified the proof to use our notation and to account for the multi-sanitizer setting. Whenever the authors of [11] refer to a message block as admissible, we say that the block is admissible for a sanitizer to which the adversary has access *i.e.,* $(\mathsf{pk}_{\mathsf{ZE}}, \mathsf{pk}_{\mathsf{ZP}}) \notin \mathbf{PKZ}^{\dagger}$. For inadmissible blocks, we say that the block is inadmissible for all sanitizers outside $\mathbf{PKZ}^{\dagger}$. The second part of the proof concerns the encryption of secret keys which is not part of the proof from [11].

To prove that k-SAN is immutable, we first show the generic hardness of the problem defined in Lemma 1.

Proof (Lemma 1). Let $(G_1^u, G_1^v, G_1^x, G_1^y, G_2^z)$ be the output of $\mathcal{A}$. Since $\mathcal{A}$ is generic, it holds that $u = u_1 + u_a a + u_b b$, $v = v_1 + v_a a + v_b b$, $x = x_1 + x_a a + x_b b$, $y = y_1 + y_a a + y_b b$, and $z = z_1 + z_b b + z_c c$ for some coefficients $u_1, u_a, u_b, v_1, v_a, v_b, x_1, x_a, x_b, y_1, y_a, y_b, z_1, z_b, z_c \in \mathbb{Z}_q$. By the relation $au - x = 0$, we have $-x_1 + (u_1 - x_a)a - x_b b + u_a a^2 + u_b ab = 0$. Note that $f(A, B) := -x_1 + (u_1 - x_a)A - x_b B + u_a A^2 + u_b AB$ is a quadratic polynomial in the variables A and B. Suppose f is not a zero polynomial, by the Schwartz-Zippel lemma (Lemma 2), for $a, b \leftarrow \mathbb{Z}_q$, the probability that $f(a, b) = 0$ is upper bounded by $2/q < 2^{1-\lambda}$ which is negligible. Therefore, we can assume that f is always zero. In particular, we have $x_1 = x_b = 0$. Similarly, by examining the relation $bv - y = 0$, we can assume that $v_1 = y_b$, and $y_1 = y_a = 0$. We can therefore write $x = x_a a$ and $y = y_b b$. Next, we examine the relation $cy - xz = 0$, which implies $y_b bc - x_a z_1 a - x_a z_b ab - x_a z_c ac = 0$. Using the Schwartz-Zippel lemma again, we can assume that $y_b = 0$. However, this means that $v = v_1 = y_b = 0$, which contradicts with the fourth relation $v \neq 0$. □

Now, suppose there exists a generic group adversary $\mathcal{A}$ against the immutability of k-SAN. We construct a generic group adversary $\mathcal{B}$ which solves the problem defined in Lemma 1. $\mathcal{B}$ receives as challenge $(G_1, G_1^a, G_1^b, G_2, G_2^c)$ from its challenger. It then simulates the $\mathsf{Exp}^{\mathsf{Immut},k}_{\text{k-SAN},\mathcal{A}}$ experiment by setting the public parameters and the signer keys honestly.

Without loss of generality, assume that $\mathcal{A}$ makes $Q_1 = \mathsf{poly}(\lambda)$ signing oracle queries and $Q_2 = \mathsf{poly}(\lambda)$ random oracle queries for $\mathcal{H}()$. $\mathcal{B}$ additionally samples $i^{\dagger}, j^{\dagger} \leftarrow [\![Q_1]\!]$ as a guess of which Sign oracle query $\mathcal{A}$ will attack against, $k^{\dagger} \leftarrow$

$[\![n]\!]$ as the index of the inadmissible block that will be modified in the forgery message, $l^\dagger \leftarrow [\![Q_2]\!]$ as a guess of which $\mathcal{H}()$ oracle query $\mathcal{A}$ will include as the inadmissible modification in the forgery message.

Answering Random Oracle Queries Upon receiving (k_l, m_l) as the l-th distinct query to the $\mathcal{H}()$ oracle, if $l \neq l^\dagger$, $\mathcal{B}$ answers the query by picking $t_l \leftarrow \mathbb{Z}_q^*$ and return $h_l := G_2^{t_l} \in \mathbb{G}_2$ to $\mathcal{A}$. If $l = l^\dagger$, then $\mathcal{B}$ sets $h_l = G_2^c$ where G_2^c was received as a challenge as described above. If (k_l, m_l) was a message that was queried previously, then reply with the same response as before.

Answering Sign Oracle Queries Upon receiving $(\mathbf{PKZ}_i, m_i, \mathbf{A}_i)$ as the i-th query to the $\mathcal{O}_{\mathsf{Sign}}$ oracle, if $i \neq i^\dagger$ and $i \neq j^\dagger$, $\mathcal{B}$ answers the query honestly by running the procedures as defined in the $\mathcal{O}_{\mathsf{Sign}}$ oracle.

In the case $i = i^\dagger$ or $i = j^\dagger$, $\mathcal{B}$ generates the signature honestly except for the following changes:

1. If $i = i^\dagger$, then $\mathcal{B}$ picks the elements $X_{i^\dagger,1}, \ldots, X_{i^\dagger,n}$ as follows. $\mathcal{B}$ picks $X_{i^\dagger,k^\dagger} = G_1^a$ which it had received from its challenger in the beginning. For all other $k \in [\![n]\!] \setminus \{k^\dagger\}$, $\mathcal{B}$ generates the X_k honestly by picking $x_{i^\dagger,k} \leftarrow \mathbb{Z}_q^*$ and setting $X_{i^\dagger,k} = G_1^{x_{i^\dagger,k}}$ (as done in the $\mathcal{O}_{\mathsf{Sign}}$ oracle). The rest of the signature is generated as in the $\mathcal{O}_{\mathsf{Sign}}$ oracle.
2. Suppose $i = j^\dagger$. If $k^\dagger$ is admissible according to $\mathbf{A}_{j^\dagger}$ for any sanitizer outside $\mathbf{PKZ}^\dagger$ *i.e.,* to which the adversary has access, or $(k^\dagger, m_{j^\dagger,k^\dagger}) = (k_{l^\dagger}, m_{l^\dagger})$, then abort. Otherwise, let $t^\dagger$ be such that $H(k^\dagger \| m_{j^\dagger,k^\dagger}) = G_2^{t^\dagger}$. $\mathcal{B}$ first generates $X_{i^\dagger,1}, \ldots, X_{i^\dagger,n}$ by picking $x_{j^\dagger,k} \leftarrow \mathbb{Z}_q^*$ and setting $X_{j^\dagger,k} := G_1^{x_{j^\dagger,k}}$ for all $k \in [\![n]\!]$. Then $\mathcal{B}$ picks the elements $Y_{i^\dagger,1}, \ldots, Y_{i^\dagger,n}$ as follows. $\mathcal{B}$ picks $Y_{j^\dagger,k^\dagger} = G_1^b$ which it had received from its challenger in the beginning. It then generates $\sigma_{j^\dagger,k^\dagger}$ as $(G_2^b)^{\frac{t^\dagger}{x_{j^\dagger,k^\dagger}}}$. For all other $k \neq k^\dagger$, $\mathcal{B}$ generates the $Y_{j^\dagger,k}$ and the rest of the signature honestly as done in the $\mathcal{O}_{\mathsf{Sign}}$ oracle. Note that as we assume $k^\dagger$ is not admissible for $\mathbf{PKZ}^* \setminus \mathbf{PKZ}^\dagger$ in this case, the value $y_{j^\dagger,k^\dagger}$ is not needed to generate the signature. Therefore, the signature can be simulated faithfully.

Clearly, assuming that $\mathcal{B}$ did not abort, $\mathcal{B}$ simulates the $\mathsf{Exp}^{\mathsf{Immut},k}_{\text{k-SAN},\mathcal{A}}$ faithfully. Eventually, $\mathcal{A}$ outputs $(\mathbf{PKZ}^*, m^*, \sigma^*)$ as a forgery such that $\mathsf{Verify}(\mathsf{pk_S}^\dagger, \mathbf{PKZ}^*, m^*, \sigma^*) = 1$, and $m_k^* \neq m_{i,k}$ for some i, k such that k is not admissible for $\mathbf{PKZ}^* \setminus \mathbf{PKZ}^\dagger$. Since $Q_1, n \in \mathsf{poly}(\lambda)$, with non-negligible probability it holds that $m_{k^\dagger}^* \neq m_{j^\dagger,k^\dagger}$ and $k^\dagger$ is not admissible for $\mathbf{PKZ}^* \setminus \mathbf{PKZ}^\dagger$. Moreover, since $Q_2 \in \mathsf{poly}(\lambda)$, with non-negligible probability it holds that $(k^\dagger, m_{k^\dagger}^*) = (k_{l^\dagger}, m_{l^\dagger})$. If that is the case, then the abort conditions in the above procedures of answering sign oracle queries are never triggered.

Parse σ^* as $(\mu^*, \eta^*, \{\sigma_j^*, X_j^*, Y_j^*\}_{j=1}^n, \mathbf{SKZ}^*)$. By the EUF-CMA-security of EQS, with overwhelming probability we have that $[X_1^*, \ldots, X_n^*]_\mathcal{R} = [X_{i^*,1}, \ldots, X_{i^*,n}]_\mathcal{R}$ for some i^*, and $[Y_1^*, \ldots, Y_n^*]_\mathcal{R} = [Y_{j^*,1}, \ldots, Y_{j^*,n}^*]_\mathcal{R}$ for some j^* (otherwise we

can construct an adversary against the EUF-CMA-security of EQS by simply outputting μ or η). Therefore, there exists $r, s \in \mathbb{Z}_q$ such that $(X_{i^*,1}, \ldots, X_{i^*,n})^r = (X_1^*, \ldots, X_n^*)$ and $(Y_{j^*,1}, \ldots, Y_{j^*,n}^*)^{r \cdot s} = (Y_1^*, \ldots, Y_n^*)$.

Since $Q_1 \in \mathsf{poly}(\lambda)$, it happens with non-negligible probability that $(i^\dagger, j^\dagger) = (i^*, j^*)$. Suppose this is the case. Let k' be arbitrary with $k' \neq k^\dagger$. $\mathcal{B}$ extracts G_1^r and G_1^{rs} by computing

$$(X_{k'}^*)^{\frac{1}{x_{i^*,k'}}} = G_1^{r \cdot \left(x_{i^*,k'}\right) \cdot \frac{1}{x_{i^*,k'}}} = G_1^r$$
$$(Y_{k'}^*)^{\frac{1}{\left(x_{j^*,k'}\right) \cdot \left(y_{j^*,k'}\right)}} = (G_1^{r \cdot s})^{\frac{\left(x_{j^*,k'}\right) \cdot \left(y_{j^*,k'}\right)}{\left(x_{j^*,k'}\right) \cdot \left(y_{j^*,k'}\right)}} = G_1^{r \cdot s}.$$

Since $\mathsf{Verify}(\mathsf{pk}_\mathsf{S}, \mathbf{PKZ}^*, m^*, \sigma^*) = 1$, this implies that $Y_{k^\dagger}^* = G_1^{r \cdot s \cdot b} \neq G_1$. This means that $r \cdot s \neq 0$. Furthermore, we have

$$\begin{aligned} e(X_{k^\dagger}^*, \sigma_{k^\dagger}^*) &= e(Y_{k^\dagger}^*, \mathcal{H}(k^\dagger \| m_{k^\dagger}^*)) \\ e(X_{i^\dagger,k^\dagger}^r, \sigma_{k^\dagger}^*) &= e(Y_{j^\dagger,k^\dagger}^{r \cdot s}, G_2^c) \\ e(G_1^{r \cdot a}, \sigma_{k^\dagger}^*) &= e(G_1^{r \cdot s \cdot b}, G_2^c) \\ \sigma_{k^\dagger}^* &= G_2^{\frac{s \cdot b \cdot c}{a}} \end{aligned}$$

Now, set $\mathcal{B}$ outputs $(G_1^u, G_1^v, G_1^x, G_1^y, G_2^z) \coloneqq (G_1^r, G_1^{r \cdot s}, G_1^{r \cdot a}, G_1^{r \cdot s \cdot b}, G_2^{\frac{s \cdot b \cdot c}{a}})$. By a routine calculation, one can verify that $au - x = 0$, $bv - y = 0$, $cy - xz = 0$ and $v \neq 0$. Since $\mathcal{A}$ only performs generic group operations, so does $\mathcal{B}$, which contradicts with Lemma 1.

The encryption is not part of Bultel *et al.*'s [11] immutability proof as the adversary is meant to have access to the signing keys of admissible blocks as he has full access to the sanitizer. Our case is different, the adversary can have access only to a subset of the admissible blocks' signing keys, *i.e.,* the message blocks that are admissible to at least one sanitizer not in $\mathbf{PKZ}^\dagger$. This means that we have another axe of attack which is breaking the encryption for a sanitizer in $\mathbf{PKZ}^\dagger$. Thus, we need to prove that PKE is IND-CPA using hybrid argument as the following. We create a hybrid called Hyb which is the same as $\mathsf{Exp}_{\text{k-SAN},\mathcal{A}}^{\mathsf{Immut},k}$, but we replace all the signing keys encrypted in $\mathbf{SKZ}$ as $\mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}_{\mathsf{ZE}}^{\mathsf{i}}, y_j)$ by $\mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}_{\mathsf{ZE}}^{\mathsf{i}}, 0)$ for all sanitizers to which the adversary does not have access. We create the hybrids $\mathsf{Hyb}_{0,n}, \mathsf{Hyb}_{1,1}, \ldots \mathsf{Hyb}_{1,n}, \ldots \mathsf{Hyb}_{k,n}$ where each $\mathsf{Hyb}_{i,j}$ differs from the previous hybrid (*i.e.,* $\mathsf{Hyb}_{i,j-1} \mid j > 1$ or $\mathsf{Hyb}_{i-1,n}$ otherwise) by that the ciphertext $\mathbf{SKZ}_{i,j}$ is an encryption of 0 instead of y_j if the adversary does not have access to the sanitizer i, *i.e.,* $(\mathsf{pk}_{\mathsf{ZE}}^{\mathsf{i}}, \mathsf{pk}_{\mathsf{ZP}}^{\mathsf{i}}) \in \mathbf{PKZ}^\dagger$. Note that $\mathsf{Hyb}_{0,n}$ is identical to $\mathsf{Exp}_{\text{k-SAN},\mathcal{A}}^{\mathsf{Immut},k}$ and $\mathsf{Hyb}_{k,n}$ is identical to Hyb. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathsf{Hyb}_{i,j}$ and the previous hybrid $\mathsf{Hyb}_{i',j'}$, we can construct an algorithm $\mathcal{B}$ that can break the IND-CPA of PKE as the following. $\mathcal{B}$ receives pk_c from the IND-CPA challenger, it will use it as the public key for the sanitizer i. The sign oracle is simulated honestly except

for the ciphertexts for the sanitizer i. For $\mathbf{SKZ}i, j$, $\mathcal{B}$ defines two messages τ_0 and τ_1 as

$$\tau_0 = \begin{cases} y_j, & a_{i,j} = 1 \\ 0, & \text{otherwise} \end{cases} \quad \tau_1 = \begin{cases} y_j, & a_{i,j} = 1 \wedge (\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}}) \notin \mathbf{PKZ}^{\dagger} \\ 0, & \text{otherwise} \end{cases}.$$

Then, $\mathcal{B}$ sends (τ_0, τ_1) to the IND-CPA challenger and gets the ciphertext c which it will use for $\mathbf{SKZ}_{i,j}$. For the other values in $\mathbf{SKZ}_i$, $\mathcal{B}$ will use the encryption oracle.

This means that the advantage of $\mathcal{A}$ to distinguish $\mathsf{Hyb}_{i,j}$ from $\mathsf{Hyb}_{i',j'}$ is equal to his advantage against the IND-CPA of PKE. Thus, the advantage of $\mathcal{A}$ to distinguish $\mathsf{Exp}^{\mathsf{Immut},k}_{\text{k-SAN},\mathcal{A}}$ from Hyb is equal to $n \cdot k \cdot \mathsf{Adv}^{\mathsf{IND-CPA}}_{\text{PKE},\mathcal{A}}(\lambda)$. As IND-CPA implies one-wayness, then we know that the adversary cannot break the encryption for sanitizers to which he does not have access.

In summary, we have shown that the adversary cannot modify inadmissible blocks as the BLS-like signatures are unforgeable (by Lemma 1), modify the public BLS-like public keys (by the EUF-CMA security of EQS), or break the encryption for sanitizers to which he does not have access (by the IND-CPA security of PKE). This gives us the following advantage

$$\mathsf{Adv}^{\mathsf{Immut},k}_{\text{k-SAN},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{Lemma\ 1}}_{\text{k-SAN},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{EUF-CMA}}_{\text{EQS},\mathcal{A}}(\lambda) + n \cdot k \cdot \mathsf{Adv}^{\mathsf{IND-CPA}}_{\text{PKE},\mathcal{A}}(\lambda)$$

which is negligible as the parameters n and k are small in real applications, and assuming the underlying schemes are secure. □

Proof (Signer accountability). We can reuse the proof of Bultel *et al.* [11]. We only modify the notation to fit our model.

Assume that there exists a polynomial time adversary $\mathcal{A}$ that breaks the signer accountability of k-SAN. We show how to build an algorithm $\mathcal{B}$ that breaks the accountability of VRS for $n = k$ where k is the number of sanitizers. The algorithm $\mathcal{B}$ receives $(\mathsf{pp}_{\mathsf{VRS}}, \{\mathsf{pk}^{\dagger,\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]})$ as input from the VRS challenger which it will use instead of the keys and public parameters generated for VRS in the k-SAN algorithms. Then, $\mathcal{B}$ runs $(\mathsf{pk}^*_{\mathsf{S}}, m^*, \sigma^*, pi^*) \leftarrow \mathcal{A}(\mathsf{pp}, \mathbf{PKZ}^{\dagger})$. While $\mathcal{A}$ is running, $\mathcal{B}$ runs the oracle $\mathcal{O}_{\mathsf{Sanit}}$ as follows.

Answering Sanitize Oracle Queries. $\mathcal{B}$ simulates the oracle honestly except when generating the VRS signature and proof. It will send $(\{\mathsf{pk}_{\mathsf{SP}}\} \cup \{\mathsf{pk}^{\dagger,\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]}, i + 1, \mathsf{pk}_{\mathsf{S}} \| m' \| \mathbf{PKZ}^{\dagger} \| \sigma'_{\mathsf{SS}})$ to the sign oracle of VRS and receives σ'_{VRS}. i here is the index of the chosen sanitizer by the adversary.

$\mathcal{B}$ parses σ^* as $(\sigma^*_{\mathsf{SS}}, \sigma^*_{\mathsf{VRS}})$ and $\mathsf{pk}^*_{\mathsf{S}}$ as $(\mathsf{pk}^*_{\mathsf{SP}}, \mathsf{pk}^*_{\mathsf{EQS}})$. It sets $L^{**} \coloneqq \{\mathsf{pk}^*_{\mathsf{SP}}\} \cup \{\mathsf{pk}^{\dagger,\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]}$, $m^{**} \coloneqq \mathsf{pk}^*_{\mathsf{S}} \| m^* \| \mathbf{PKZ}^{\dagger} \| \sigma^*_{\mathsf{SS}}$, $\sigma^{**} \coloneqq \sigma^*_{\mathsf{VRS}}$, $\mathsf{pk}^{**} \coloneqq \mathsf{pk}^*_{\mathsf{SP}}$ and $\pi^{**} \coloneqq \pi^*$. Finally, $\mathcal{B}$ returns $(L^{**}, m^{**}, \sigma^{**}, \mathsf{pk}^{**}, \pi^{**})$.

Clearly, the experiment is perfectly simulated for $\mathcal{A}$. Assume that $\mathcal{A}$ wins its experiment, the following holds:

- $((\mathsf{pk}^*_{\mathsf{S}}, m^*, \sigma^*) \notin \{(\mathsf{pk}^i_{\mathsf{S}}, m_i, \sigma_i)\}_{i=1}^{|\Sigma|})$. Parse σ^* as $(\sigma^*_{\mathsf{SS}}, \sigma^*_{\mathsf{VRS}})$. By rearranging terms we have, $((\mathsf{pk}^*_{\mathsf{S}}, \mathbf{PKZ}^\dagger, m^*, \sigma^*_{\mathsf{SS}}), \sigma^*_{\mathsf{VRS}}) \neq ((\mathsf{pk}^i_{\mathsf{S}}, \mathbf{PKZ}^\dagger, m_i, \sigma_{\mathsf{SS},i}), \sigma_{\mathsf{VRS},i})$ for each i.
- $\mathsf{Verify}(\mathsf{pk}^*_{\mathsf{S}}, \mathbf{PKZ}^\dagger, m^*, \sigma^*) = 1$ which implies that $\mathsf{Verify}_{\mathsf{VRS}}(L^{**}, \sigma^{**}, m^{**}) = 1$.
- $\mathsf{Judge}(\mathsf{pk}^*_{\mathsf{S}}, \mathbf{PKZ}^\dagger, m^*, \sigma^*, \pi^*, \perp) \neq S$ which implies that $\mathsf{Judge}_{\mathsf{VRS}}(L^{**}, m^{**}, \sigma^{**}, \mathsf{pk}^{**}, \pi^{**}) = 0$.

Finally, note that $L^{**} \subseteq (\{\mathsf{pk}^{\dagger,\mathsf{i}}_{\mathsf{ZP}}\}_{i\in[\![k]\!]} \cup \{\mathsf{pk}^{**}\})$. Therefore, $\mathsf{Exp}^{\mathsf{Acc},k}_{\mathrm{VRS},\mathcal{B}}(\lambda)$ returns 1 with at least the probability that $\mathsf{Exp}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$ returns 1, which is a contradiction. This concludes the proof giving the following advantage which is negligible assuming that VRS is secure.

$$\mathsf{Adv}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{Acc},k}_{\mathrm{VRS},\mathcal{A}}(\lambda)$$

□

Proof (Sanitizer accountability). We can reuse the proof of Bultel *et al.* [11]. We only modify the notation to fit our model.

Assume that there exists a polynomial time adversary $\mathcal{A}$ that breaks the sanitizer accountability of k-SAN. We show how to build an algorithm $\mathcal{B}$ that breaks the non-seizability of VRS. The algorithm $\mathcal{B}$ receives $(\mathsf{pp}_{\mathsf{VRS}}, \mathsf{pk}^\dagger_{\mathsf{SP}})$ as input from the VRS challenger which it will use instead of the signer key and public parameters generated for VRS in the k-SAN algorithms. Then, $\mathcal{B}$ runs $(\mathbf{PKZ}^*, m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}^\dagger_{\mathsf{S}})$. While $\mathcal{A}$ is running, $\mathcal{B}$ runs the signing oracle $\mathcal{O}_{\mathsf{Sign}}$ as follows.

Answering Sign Oracle Queries. $\mathcal{B}$ simulates the oracle honestly except when generating the VRS signature. It will send $(\{\mathsf{pk}^\dagger_{\mathsf{SP}}\} \cup \{\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}}\}_{i\in[\![k]\!]}, 1, \mathsf{pk}^\dagger_{\mathsf{S}} \| m \| \mathbf{PKZ} \| \sigma_{\mathsf{SS}})$ to the sign oracle of VRS and receive σ_{VRS}.

Answering Prove Oracle Queries. $\mathcal{B}$ will send $(\{\mathsf{pk}^\dagger_{\mathsf{SP}}\} \cup \{\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}}\}_{i\in[\![k]\!]}, \mathsf{pk}^\dagger_{\mathsf{S}} \| m \| \mathbf{PKZ} \| \sigma_{\mathsf{SS}}, 1, \sigma_{\mathsf{VRS}})$ to the prove oracle of VRS and get the π' from it.

$\mathcal{B}$ parses σ^* as $(\sigma^*_{\mathsf{SS}}, \sigma^*_{\mathsf{VRS}})$ and $\mathbf{PKZ}^*$ as $\{(\mathsf{pk}^{*,\mathsf{i}}_{\mathsf{ZE}}, \mathsf{pk}^{*,\mathsf{i}}_{\mathsf{ZP}})\}_{i\in[\![k]\!]}$. It sets $L^{**} \coloneqq \{\mathsf{pk}^\dagger_{\mathsf{SP}}\} \cup \{\mathsf{pk}^{*,\mathsf{i}}_{\mathsf{ZP}}\}_{i\in[\![k]\!]}$, $m^{**} \coloneqq \mathsf{pk}^\dagger_{\mathsf{S}} \| m^* \| \mathbf{PKZ}^* \| \sigma^*_{\mathsf{SS}}$, and $\sigma^{**} \coloneqq \sigma^*_{\mathsf{VRS}}$. Finally, $\mathcal{B}$ returns $(L^{**}, m^{**}, \sigma^{**})$.

Clearly, the experiment is perfectly simulated for $\mathcal{A}$. Assume that $\mathcal{A}$ wins its experiment, the following holds:

- $((\mathbf{PKZ}^*, m^*, \sigma^*) \notin \{((\mathbf{PKZ}_i, m_i, \sigma_i))\}_{i=1}^{|\Sigma|})$. Parse σ^* as $(\sigma^*_{\mathsf{SS}}, \sigma^*_{\mathsf{VRS}})$. By rearranging terms we have, $((\mathsf{pk}^\dagger_{\mathsf{S}}, \mathbf{PKZ}^*, m^*, \sigma^*_{\mathsf{SS}}), \sigma^*_{\mathsf{VRS}}) \neq ((\mathsf{pk}^\dagger_{\mathsf{S}}, \mathbf{PKZ}_i, m_i, \sigma_{\mathsf{SS},i}), \sigma_{\mathsf{VRS},i})$ for each i.

- $\mathsf{Verify}(\mathsf{pk}_{\mathsf{S}}^{\dagger}, \mathbf{PKZ}^*, m^*, \sigma^*) = 1$ which implies that $\mathsf{Verify}_{\mathsf{VRS}}(L^{**}, \sigma^{**}, m^{**}) = 1$.
- $\mathsf{Judge}(\mathsf{pk}_{\mathsf{S}}^{\dagger}, \mathbf{PKZ}^*, m^*, \sigma^*, \pi^*, \perp) \neq Z$ which implies that $\mathsf{Judge}_{\mathsf{VRS}}(L^{**}, m^{**}, \sigma^{**}, \mathsf{pk}^{**}, \pi) \neq 0$ for any $\pi^* \leftarrow \mathsf{Prove}(\mathsf{sk}_{\mathsf{S}}^{\dagger}, \mathbf{PKZ}^*, m^*, \sigma^*, \pi^*, \perp)$.

Therefore, $\mathsf{Exp}^{\mathsf{Non-Seiz}}_{\mathrm{VRS},\mathcal{B}}(\lambda)$ returns 1 with at least the probability that $\mathsf{Exp}^{\mathsf{SanAcc}}_{\text{k-SAN},\mathcal{A}}(\lambda)$ returns 1, which is a contradiction. This concludes the proof giving the following advantage which is negligible assuming that VRS is secure.

$$\mathsf{Adv}^{\mathsf{SanAcc}}_{\text{k-SAN},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{Non-Seiz}}_{\mathrm{VRS},\mathcal{A}}(\lambda)$$

□

Proof (Proof-Restricted Transparency). We use the same proof as Bultel *et al.* [11], but we adapt the notation to our model and add two new distributions $\mathcal{D}'''$ and $\mathcal{D}''''$ for the ciphertexts and the VRS signature.

We show that the construction is perfectly strongly transparent through hybrid argument.

First, let $Q = \mathsf{poly}(\lambda)$ be the number of queries that the adversary $\mathcal{A}$ made to the $\mathcal{O}^b_{\mathsf{Sign/Sanit}}$ oracle. We define the hybrids $\mathsf{Hyb}_0, \ldots, \mathsf{Hyb}_q$ as follows. The hybrid Hyb_0 is identical to $\mathsf{Exp}^{\mathsf{Trans},k,0}_{\text{k-SAN},\mathcal{A}}(\lambda)$. For $l \in [Q]$, Hyb_l is almost identical to Hyb_{l-1}, except that the l-th query to the $\mathcal{O}^b_{\mathsf{Sign/Sanit}}$ is answered as in $\mathsf{Exp}^{\mathsf{Trans},k,1}_{\text{k-SAN},\mathcal{A}}(\lambda)$. That is, the first l signatures returned by $\mathcal{O}^b_{\mathsf{Sign/Sanit}}$ are freshly signed, while the last $Q-l$ signatures are sanitized. Note that Hyb_Q is identical to $\mathsf{Exp}^{\mathsf{Trans},k,1}_{\text{k-SAN},\mathcal{A}}(\lambda)$. Obviously, if $\mathsf{Pr}\left[\mathsf{Hyb}_{lj-1} = 1\right] = \mathsf{Pr}\left[\mathsf{Hyb}_l = 1\right]$ for all $l \in [Q]$, then $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Trans},k,0}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] = \mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{Trans},k,1}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right]$.

Fix $l \in [Q]$. In the following, we show that $\mathsf{Pr}\left[\mathsf{Hyb}_{l-1} = 1\right] = \mathsf{Pr}\left[\mathsf{Hyb}_l = 1\right]$. Let $(\mathsf{pk}_{\mathsf{Z}}^{\dagger,i}, m, \mathsf{MOD}, \mathbf{A})$ be the l-th query of $\mathcal{A}$ to the $\mathcal{O}^b_{\mathsf{Sign/Sanit}}$ oracle. The oracle returns $\perp$ in both experiments if $\neg\mathsf{ADM}^{\mathbf{A}}_{\mathsf{SAN}}(m, \mathsf{MOD}(m), i)$, and thus the equality holds trivially. Otherwise, let $m' \coloneqq \mathsf{MOD}(m)$, and let σ' be the response. In Hyb_{l-1}, the signature σ' is drawn from a distribution $\mathcal{D}$ where

$$\mathcal{D} \coloneqq \left\{ \sigma : \begin{array}{l} x_j, y_j \leftarrow \mathbb{Z}_q^*,\ X_j \coloneqq G_1^{x_j},\ Y_j \coloneqq X_j^{y_j},\ \forall j \in [\![n]\!] \\ \mu \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (X_1, \ldots, X_n)) \\ \eta \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (Y_1, \ldots, Y_n)) \\ \sigma_j \leftarrow \mathcal{H}(j \| m'_j)^{y_j},\ \forall j \in [\![n]\!] \\ \mathbf{SKZ}_{i,j} \leftarrow \mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, y_j \cdot a_{i,j}), \forall i \in [\![k]\!], j \in [\![n]\!] \\ \sigma_{\mathsf{SS}} \coloneqq (\mu, \eta, \{\sigma_j, X_j, Y_j\}_j^n, \mathbf{SKZ}) \\ t \coloneqq \mathsf{pk}^{\dagger}_{\mathsf{S}} \| m \| \mathbf{PKZ}^{\dagger} \| \sigma_{\mathsf{SS}}, L \coloneqq \{\mathsf{pk}^{\dagger}_{\mathsf{SP}}\} \cup \{\mathsf{pk}^{\dagger,\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]} \\ \sigma_{\mathsf{VRS}} \leftarrow \mathsf{Sign}_{\mathsf{VRS}}(\mathsf{sk}_{\mathsf{SP}}, L, t) \\ \sigma \coloneqq (\sigma_{\mathsf{SS}}, \sigma_{\mathsf{VRS}}) \end{array} \right\}.$$

Replacing x_i and y_i with $r \cdot x_i$ and $s \cdot y_i$ respectively for some $r, s \leftarrow \mathbb{Z}_q^*$, we obtain a distribution $\mathcal{D}' = \mathcal{D}$ where

$$\mathcal{D}' \coloneqq \left\{ \sigma : \begin{array}{l} r, s \leftarrow \mathbb{Z}_q^* \\ x_j, y_j \leftarrow \mathbb{Z}_q^*,\ X_j \coloneqq G_1^{x_j},\ Y_j \coloneqq X_j^{y_j},\ \forall j \in [\![n]\!] \\ \mu \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (X_1, \ldots, X_n)^r) \\ \eta \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (Y_1, \ldots, Y_n)^{r \cdot s}) \\ \sigma_j \leftarrow \mathcal{H}(j \| m'_j)^{s \cdot y_j},\ \forall j \in [\![n]\!] \\ \mathbf{SKZ}_{i,j} \leftarrow \mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, s \cdot y_j \cdot a_{i,j}), \forall i \in [\![k]\!], j \in [\![n]\!] \\ \sigma_{\mathsf{SS}} \coloneqq (\mu, \eta, \{\sigma_j, X_j^r, Y_j^{r \cdot s}\}_j^n, \mathbf{SKZ}) \\ t \coloneqq \mathsf{pk}^{\dagger}_{\mathsf{S}} \| m \| \mathbf{PKZ}^{\dagger} \| \sigma_{\mathsf{SS}}, L \coloneqq \{\mathsf{pk}^{\dagger}_{\mathsf{SP}}\} \cup \{\mathsf{pk}^{\dagger,\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]} \\ \sigma_{\mathsf{VRS}} \leftarrow \mathsf{Sign}_{\mathsf{VRS}}(\mathsf{sk}_{\mathsf{SP}}, L, t) \\ \sigma \coloneqq (\sigma_{\mathsf{SS}}, \sigma_{\mathsf{VRS}}) \end{array} \right\}.$$

By the perfect adaption of EQS, the distribution of $\mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (X_1, \ldots, X_n)^r)$ and $\mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (Y_1, \ldots, Y_n)^{r \cdot s})$ is identical to that of $\mathsf{ChgRep}_{\mathsf{EQS}}(\mathsf{pk}^{\dagger}_{\mathsf{EQS}}, (X_1, \ldots, X_n), \mu', r)$ and $\mathsf{ChgRep}_{\mathsf{EQS}}(\mathsf{pk}^{\dagger}_{\mathsf{EQS}}, (Y_1, \ldots, Y_n), \eta', r \cdot s)$, where $\mu' \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (X_1, \ldots, X_n))$ and $\eta' \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (Y_1, \ldots, Y_n))$. Therefore, we obtain a distribution $\mathcal{D}'' = \mathcal{D}'$ with

$$\mathcal{D}'' \coloneqq \left\{ \sigma : \begin{array}{l} r, s \leftarrow \mathbb{Z}_q^* \\ x_j, y_j \leftarrow \mathbb{Z}_q^*,\ X_j \coloneqq G_1^{x_j},\ Y_j \coloneqq X_j^{y_j},\ \forall j \in [\![n]\!] \\ \mu' \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (X_1, \ldots, X_n)) \\ \eta' \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (Y_1, \ldots, Y_n)) \\ \mu \leftarrow \mathsf{ChgRep}_{\mathsf{EQS}}(\mathsf{pk}^{\dagger}_{\mathsf{EQS}}, (X_1, \ldots, X_n), \mu', r) \\ \eta \leftarrow \mathsf{ChgRep}_{\mathsf{EQS}}(\mathsf{pk}^{\dagger}_{\mathsf{EQS}}, (Y_1, \ldots, Y_n), \eta', r \cdot s) \\ \sigma_j \leftarrow \mathcal{H}(j \| m'_j)^{s \cdot y_j},\ \forall j \in [\![n]\!] \\ \mathbf{SKZ}_{i,j} \leftarrow \mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, s \cdot y_j \cdot a_{i,j}), \forall i \in [\![k]\!], j \in [\![n]\!] \\ \sigma_{\mathsf{SS}} \coloneqq (\mu, \eta, \{\sigma_j, X_j^r, Y_j^{r \cdot s}\}_j^n, \mathbf{SKZ}) \\ t \coloneqq \mathsf{pk}^{\dagger}_{\mathsf{S}} \| m \| \mathbf{PKZ}^{\dagger} \| \sigma_{\mathsf{SS}}, L \coloneqq \{\mathsf{pk}^{\dagger}_{\mathsf{SP}}\} \cup \{\mathsf{pk}^{\dagger,\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]} \\ \sigma_{\mathsf{VRS}} \leftarrow \mathsf{Sign}_{\mathsf{VRS}}(\mathsf{sk}_{\mathsf{SP}}, L, t) \\ \sigma \coloneqq (\sigma_{\mathsf{SS}}, \sigma_{\mathsf{VRS}}) \end{array} \right\}.$$

As PKE is unlinkable, the distribution of $\mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, s \cdot y_j \cdot a_{i,j})$ is identical to that of $\mathsf{Multiply}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathbf{SKZ}'_{i,j}, s)$ where $\mathbf{SKZ}'_{i,j} \leftarrow \mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, y_j \cdot$

$a_{i,j}$). Thus, we obtain a distribution $\mathcal{D}''' = \mathcal{D}''$.

$$\mathcal{D}''' \coloneqq \left\{ \sigma : \begin{array}{l} r, s \leftarrow \mathbb{Z}_q^* \\ x_j, y_j \leftarrow \mathbb{Z}_q^*,\ X_j \coloneqq G_1^{x_j},\ Y_j \coloneqq X_j^{y_j},\ \forall j \in [\![n]\!] \\ \mu' \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (X_1, \ldots, X_n)) \\ \eta' \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (Y_1, \ldots, Y_n)) \\ \mu \leftarrow \mathsf{ChgRep}_{\mathsf{EQS}}(\mathsf{pk}^{\dagger}_{\mathsf{EQS}}, (X_1, \ldots, X_n), \mu', r) \\ \eta \leftarrow \mathsf{ChgRep}_{\mathsf{EQS}}(\mathsf{pk}^{\dagger}_{\mathsf{EQS}}, (Y_1, \ldots, Y_n), \eta', r \cdot s) \\ \sigma_j \leftarrow \mathcal{H}(j \| m'_j)^{s \cdot y_j},\ \forall j \in [\![n]\!] \\ \mathbf{SKZ}'_{i,j} \leftarrow \mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, y_j \cdot a_{i,j}), \forall i \in [\![k]\!], j \in [\![n]\!] \\ \mathbf{SKZ}_{i,j} \leftarrow \mathsf{Multiply}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathbf{SKZ}'_{i,j}, s), \forall i \in [\![k]\!], j \in [\![n]\!] \\ \sigma_{\mathsf{SS}} \coloneqq (\mu, \eta, \{\sigma_j, X_j^r, Y_j^{r \cdot s}\}_j^n, \mathbf{SKZ}) \\ t \coloneqq \mathsf{pk}^{\dagger}_{\mathsf{S}} \| m \| \mathbf{PKZ}^{\dagger} \| \sigma_{\mathsf{SS}}, L \coloneqq \{\mathsf{pk}^{\dagger}_{\mathsf{SP}}\} \cup \{\mathsf{pk}^{\dagger,\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]} \\ \sigma_{\mathsf{VRS}} \leftarrow \mathsf{Sign}_{\mathsf{VRS}}(\mathsf{sk}_{\mathsf{SP}}, L, t) \\ \sigma \coloneqq (\sigma_{\mathsf{SS}}, \sigma_{\mathsf{VRS}}) \end{array} \right\}.$$

As VRS is anonymous, the distribution of $\mathsf{Sign}_{\mathsf{VRS}}(\mathsf{sk}_{\mathsf{ZP}}, L, t)$ is identical to that of $\mathsf{Sign}_{\mathsf{VRS}}(\mathsf{sk}_{\mathsf{SP}}, L, t)$. Thus, we obtain a distribution $\mathcal{D}'''' = \mathcal{D}'''$.

$$\mathcal{D}'''' \coloneqq \left\{ \sigma : \begin{array}{l} r, s \leftarrow \mathbb{Z}_q^* \\ x_j, y_j \leftarrow \mathbb{Z}_q^*,\ X_j \coloneqq G_1^{x_j},\ Y_j \coloneqq X_j^{y_j},\ \forall j \in [\![n]\!] \\ \mu' \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (X_1, \ldots, X_n)) \\ \eta' \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (Y_1, \ldots, Y_n)) \\ \mu \leftarrow \mathsf{ChgRep}_{\mathsf{EQS}}(\mathsf{pk}^{\dagger}_{\mathsf{EQS}}, (X_1, \ldots, X_n), \mu', r) \\ \eta \leftarrow \mathsf{ChgRep}_{\mathsf{EQS}}(\mathsf{pk}^{\dagger}_{\mathsf{EQS}}, (Y_1, \ldots, Y_n), \eta', r \cdot s) \\ \sigma_j \leftarrow \mathcal{H}(j \| m'_j)^{s \cdot y_j},\ \forall j \in [\![n]\!] \\ \mathbf{SKZ}'_{i,j} \leftarrow \mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, y_j \cdot a_{i,j}), \forall i \in [\![k]\!], j \in [\![n]\!] \\ \mathbf{SKZ}_{i,j} \leftarrow \mathsf{Multiply}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathbf{SKZ}'_{i,j}, s), \forall i \in [\![k]\!], j \in [\![n]\!] \\ \sigma_{\mathsf{SS}} \coloneqq (\mu, \eta, \{\sigma_j, X_j^r, Y_j^{r \cdot s}\}_j^n, \mathbf{SKZ}) \\ t \coloneqq \mathsf{pk}^{\dagger}_{\mathsf{S}} \| m \| \mathbf{PKZ}^{\dagger} \| \sigma_{\mathsf{SS}}, L \coloneqq \{\mathsf{pk}^{\dagger}_{\mathsf{SP}}\} \cup \{\mathsf{pk}^{\dagger,\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]} \\ \sigma_{\mathsf{VRS}} \leftarrow \mathsf{Sign}_{\mathsf{VRS}}(\mathsf{sk}_{\mathsf{ZP}}, L, t) \\ \sigma \coloneqq (\sigma_{\mathsf{SS}}, \sigma_{\mathsf{VRS}}) \end{array} \right\}.$$

Note that in Hyb_j, the signature σ' is drawn exactly from $\mathcal{D}''''$. Therefore, we can conclude that Hyb_{j-1} and Hyb_j are functionally equivalent. This yields the following advantage against proof-restricted transparency

$$\mathsf{Adv}^{\mathsf{Trans},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{Per-Adapt}}_{\mathrm{EQS},\mathcal{A}}(\lambda) + n \cdot k \cdot \mathsf{Adv}^{\mathsf{Unlink}}_{\mathrm{PKE},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{Anon},k+1}_{\mathrm{VRS},\mathcal{A}}(\lambda)$$

which is negligible because n and k are small in real applications, and assuming the underlying schemes are secure. □

Proof (Invisibility). We use the same proof idea as Bultel *et al.* [11], but we adapt the notation and modify their proof to fit the structure of our ciphertexts.

We prove invisibility by hybrid argument. We define an intermediate experiment Hyb^b which is identical to $\mathsf{Exp}^{\mathsf{Inv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$ for both $b \in \{0,1\}$, except for the following changes: When answering $\mathcal{O}^b_{\mathsf{LoRADM}}$ oracle queries, the challenger signs with respect to the policy $\mathbf{A}_0 \wedge \mathbf{A}_1$ instead of $\mathbf{A}_b$. We argue that, in the view of the adversary, the experiments Hyb^b and $\mathsf{Exp}^{\mathsf{Inv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$ are computationally indistinguishable for $b \in \{0,1\}$. Suppose that this is the case, since obviously Hyb^0 is functionally equivalent to Hyb^1, it holds that $\mathsf{Exp}^{\mathsf{Inv},k,0}_{\text{k-SAN},\mathcal{A}}(\lambda)$ is computationally indistinguishable to $\mathsf{Exp}^{\mathsf{Inv},k,1}_{\text{k-SAN},\mathcal{A}}(\lambda)$.

Before proving the claim above, we state two key observations. First, note that the signatures returned by the $\mathcal{O}^b_{\mathsf{LoRADM}}$ oracle in the all experiments are identically distributed if $\mathbf{PKZ} \neq \mathbf{PKZ}^\dagger$ (since now it must hold that $\mathbf{A}_0 = \mathbf{A}_1$ for the oracle to not abort). In the case $\mathbf{PKZ} = \mathbf{PKZ}^\dagger$, the signatures returned by the oracle are almost identically distributed, except for the ciphertexts matrix $\mathbf{SKZ}$. In particular, in the experiment $\mathsf{Exp}^{\mathsf{Inv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$, each ciphertext $\mathbf{SKZ}_{b,i,j}$ is an encryption of the message $\zeta_{b,i,j}$, where $\zeta_{b,i,j} = y_j$ for all $i \in [\![k]\!], j \in [\![n]\!] \mid a_{b,i,j}$, and is 0 otherwise. On the other hand, in Hyb^b, each ciphertext $\mathbf{SKZ}_{b,i,j}$ is an encryption of the message $\zeta_{b,i,j}$, where $\zeta_{b,i,j} = y_j$ for all $i \in [\![k]\!], j \in [\![n]\!] \mid a_{0,i,j} \wedge a_{1,i,j}$, and is 0 otherwise.

The second observation is that, due to the restriction imposed on the $\mathcal{O}_{\mathsf{Sanit}'}$ oracle, the values y_j for all $j \in [\![n]\!] \mid a_{b,i,j} \wedge \neg(a_{0,i,j} \wedge a_{1,i,j})$ are never used in any experiments.

With the above observations, we build additional intermediate hybrids, $\mathsf{Hyb}^b_{0,n}$, $\mathsf{Hyb}^b_{1,1}, \ldots, \mathsf{Hyb}^b_{1,n}, \ldots, \mathsf{Hyb}^b_{k,n}$ where each $\mathsf{Hyb}^b_{i,j}$ differs from the previous hybrid (*i.e.,* $\mathsf{Hyb}^b_{i,j-1} \mid j > 1$ or $\mathsf{Hyb}^b_{i-1,n}$ otherwise) by that the ciphertext $\mathbf{SKZ}_{i,j}$ is an encryption of the message $\zeta_{b,i,j}$, where $\zeta_{b,i,j} = y_j$ if $a_{0,i,j} \wedge a_{1,i,j}$, and is 0 otherwise, instead of, $\zeta_{b,i,j} = y_j$ if $a_{b,i,j}$, and is 0 otherwise. Note that $\mathsf{Hyb}^b_{0,n}$ is identical to $\mathsf{Exp}^{\mathsf{Inv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$ and $\mathsf{Hyb}^b_{k,n}$ is identical to Hyb^b.

Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathsf{Hyb}^b_{i,j}$ and the previous hybrid $\mathsf{Hyb}^b_{i',j'}$, we can construct an algorithm $\mathcal{B}$ that can break the IND-CPA of PKE as the following. $\mathcal{B}$ receives pk_c from the IND-CPA challenger, it will use it as the public key for the sanitizer i.

Answering $\mathcal{O}^b_{\mathsf{LoRADM}}$ Queries. The oracle is simulated honestly except for the ciphertexts for the sanitizer i. For $\mathbf{SKZ}i, j$, $\mathcal{B}$ defines two messages τ_0 and τ_1 as

$$\tau_0 = \begin{cases} y_{i,j}, & a_{b,i,j} \\ 0, & \text{otherwise} \end{cases} \qquad \tau_1 = \begin{cases} y_{i,j}, & a_{0,i,j} \wedge a_{1,i,j} \\ 0, & \text{otherwise} \end{cases}.$$

Then, $\mathcal{B}$ sends (τ_0, τ_1) to the IND-CPA challenger and gets the ciphertext c which it will use for $\mathbf{SKZ}_{i,j}$. For the other values in $\mathbf{SKZ}_i$, $\mathcal{B}$ will use the encryption

oracle. $\mathcal{B}$ will also maintain history of the values of $y_{i,j}$ where $a_{0,i,j} \wedge a_{1,i,j}$ in order to answer calls to the $\mathcal{O}_{\mathsf{Sanit}'}$.

Answering $\mathcal{O}_{\mathsf{Sanit}'}$ Queries. Instead of decrypting the values of $\mathbf{SKZ}$, $\mathcal{B}$ will use its history that it maintained in the $\mathcal{O}^b_{\mathsf{LoRADM}}$ queries to do the sanitization. It should also maintain history of the modified values of $y_{i,j}$ in order to be able to do multiple consecutive sanitizations on the same signature.

Answering $\mathcal{O}_{\mathsf{Prove}}$ Queries. The oracle is answered honestly.

This means that the advantage of $\mathcal{A}$ to distinguish $\mathsf{Hyb}^b_{i,j}$ from $\mathsf{Hyb}^b_{i',j'}$ is equal to his advantage against the IND-CPA of PKE. This yields the following advantage

$$\mathsf{Adv}^{\mathsf{Inv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = n \cdot k \cdot \mathsf{Adv}^{\mathsf{IND-CPA}}_{\text{PKE},\mathcal{A}}(\lambda)$$

which is negligible because n and k are small in real applications and assuming that PKE is secure. □

Proof (Unlinkability). We use the same proof as Bultel *et al.* [11], but we adapt the notation to our model and reorder the proof as the following. Bultel *et al.* [11] defined a relaxed notion of unlinkability called weak unlinkability where the adversary is only allowed to query the $\mathcal{O}^b_{\mathsf{LoRSanit}}$ oracle on honestly generated signatures. They proved that Π_1 is weakly unlinkable and then proved that Π_2 is unlinkable because of VRS's unforgeability and Π_1's weak unlinkability. In our construction's proof, we will first start by proving the unforgeability of VRS then the unlinkability of k-SAN.

We define the following sequence of hybrid experiments:

Hyb^b_0: is identical to $\mathsf{Exp}^{\mathsf{Unlink},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$.

Hyb^b_1: is identical to Hyb^b_0 except for the following change. Let $(\mathsf{pk}^{\dagger,i_0}_{\mathsf{Z}}, m_0, \mathsf{MOD}_0, \sigma_0, \mathsf{pk}^{\dagger,i_1}_{\mathsf{Z}}, m_1, \mathsf{MOD}_1, \sigma_1)$ be a query from $\mathcal{A}$ to $\mathcal{O}^b_{\mathsf{LoRSanit}}$. Suppose that $\mathsf{Verify}(\mathsf{pk}^{\dagger}_{\mathsf{S}}, \mathbf{PKZ}^{\dagger}, m_\beta, \sigma_\beta) = 1$ for all $\beta \in \{0,1\}$. Then if for some $\beta \in \{0,1\}$, there is no $\mathbf{A}_\beta$ which satisfies $(\mathsf{pk}^{\dagger}_{\mathsf{S}}, \mathbf{PKZ}^{\dagger}, m_\beta, \mathbf{A}_\beta, \sigma_{\mathsf{SS},\beta}) \in \Sigma$, the challenger aborts.

Lemma 3. *If* VRS *is unforgeable, then the probability of the challenger aborting is negligible, which implies*

$$\left|\mathsf{Pr}\left[\mathsf{Hyb}^b_0 = 1\right] - \mathsf{Pr}\left[\mathsf{Hyb}^b_1 = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Suppose that there exists a polynomial time adversary $\mathcal{A}$ that forces the challenger to abort in Hyb^b_1 with non-negligible probability, we show how to build a polynomial time adversary $\mathcal{B}$ that breaks the unforgeability of the VRS scheme with non-negligible probability. $\mathcal{B}$ receives the set of public keys $\{\mathsf{pk}_{\mathsf{SP}}\} \cup \{\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]}$ and the public parameter $\mathsf{pp}_{\mathsf{VRS}}$ which it will use instead of the VRS

public keys generated by the k-SAN algorithms. It then runs $\mathcal{A}(\mathsf{pp}, \mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger)$. While $\mathcal{A}$ is running, $\mathcal{B}$ simulates the oracles $\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}_{\mathsf{Prove}}$ and $\mathcal{O}^b_{\mathsf{LoRSanit}}$ as follows:

Answering $\mathcal{O}_{\mathsf{Sign}}$ Queries. $\mathcal{B}$ simulates this oracle honestly except that it generates σ_{VRS} using the sign oracle of VRS on the input $(\{\mathsf{pk}_\mathsf{SP}\} \cup \{\mathsf{pk}^i_\mathsf{ZP}\}_{i\in[\![k]\!]}, 1, \mathsf{pk}_\mathsf{S} \| m \| \mathbf{PKZ} \| \sigma_\mathsf{SS})$.

Answering $\mathcal{O}_{\mathsf{Sanit}}$ Queries. $\mathcal{B}$ simulates this oracle honestly except that it generates σ_{VRS} by the sign oracle of VRS on $(\{\mathsf{pk}_\mathsf{SP}\} \cup \{\mathsf{pk}^i_\mathsf{ZP}\}_{i\in[\![k]\!]}, i{+}1, \mathsf{pk}_\mathsf{S} \| \mathsf{MOD}(m) \| \mathbf{PKZ} \| \sigma_\mathsf{SS}')$ where i is the index of the sanitizer that did the modification.

Answering $\mathcal{O}_{\mathsf{Prove}}$ Queries. $\mathcal{B}$ simulates this oracle honestly except that it generates the proof π by calling the prove oracle of VRS on $(\{\mathsf{pk}_\mathsf{SP}\} \cup \{\mathsf{pk}^i_\mathsf{ZP}\}_{i\in[\![k]\!]}, 1, \mathsf{pk}_\mathsf{S} \| m \| \mathbf{PKZ} \| \sigma_\mathsf{SS}, \sigma_\mathsf{VRS})$.

Answering $\mathcal{O}^b_{\mathsf{LoRSanit}}$ Queries. Let $(\mathsf{pk}_\mathsf{Z}^{\dagger,i_0}, m_0, \mathsf{MOD}_0, \sigma_0, \mathsf{pk}_\mathsf{Z}^{\dagger,i_1}, m_1, \mathsf{MOD}_1, \sigma_1)$ be a query from $\mathcal{A}$. Suppose that $\mathsf{Verify}(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m_\beta, \sigma_\beta) = 1$ for all $\beta \in \{0,1\}$ (otherwise the oracle outputs $\perp$). This implies that

$$\mathsf{Verify}_\mathsf{VRS}(\{\mathsf{pk}_\mathsf{SP}\} \cup \{\mathsf{pk}^i_\mathsf{ZP}\}, \mathsf{pk}_\mathsf{S}^\dagger \| m_\beta \| \mathbf{PKZ}^\dagger \| \sigma_{\mathsf{SS},\beta}, \sigma_{\mathrm{VRS},\beta}) = 1.$$

Suppose further that for some $\beta \in \{0,1\}$, there is no $\mathbf{A}_\beta$ which satisfies $(\mathsf{pk}_\mathsf{S}^\dagger, \mathbf{PKZ}^\dagger, m_\beta, \mathbf{A}_\beta, \sigma_\beta) \in \Sigma$. Then $\mathcal{B}$ aborts the simulation and outputs $(\{\mathsf{pk}_\mathsf{SP}\} \cup \{\mathsf{pk}^i_\mathsf{ZP}\}_{i\in[\![k]\!]}, \mathsf{pk}_\mathsf{S}^\dagger \| m_\beta \| \mathbf{PKZ}^\dagger \| \sigma_{\mathsf{SS},\beta}, \sigma_{\mathrm{VRS},\beta})$ as a forgery. Otherwise, $\mathcal{B}$ simulates this oracle honestly except that it generates σ_{VRS} by calling the sign oracle of VRS on $(\{\mathsf{pk}_\mathsf{SP}\} \cup \{\mathsf{pk}^i_\mathsf{ZP}\}, i{+}1, \mathsf{pk}_\mathsf{S}^\dagger \| \mathsf{MOD}_b(m_b) \| \mathbf{PKZ}^\dagger \| \sigma_{\mathsf{SS},b})$, where $\sigma_b = (\sigma_{\mathsf{SS},b}, \sigma_{\mathrm{VRS},b})$ and i is the index of the sanitizer used based on the value of b.

Clearly, if $\mathcal{B}$ aborts and returns a forgery, then it breaks the unforgeability of VRS. We can therefore assume that $\mathcal{B}$ does not abort with overwhelming probability. □

As we have proven that the adversary cannot forge signatures, the following proof, proves the unlinkability rather than the weak unlinkability of k-SAN.

To show that the experiments $\mathsf{Exp}^{\mathsf{Unlink},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$, where $b \in \{0,1\}$, are computationally indistinguishable in the view of the adversary, we define the following sequence of hybrids.

Hyb^b_0 : Defined as $\mathsf{Exp}^{\mathsf{Unlink},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda)$.

Hyb^b_1 : Defined as Hyb^b_0, except that a history is kept of all ciphertexts in $\mathbf{SKZ}$ and their corresponding plaintexts generated using PKE. This is done as the following. An additional list $\tilde{L}$ is initialized empty at the beginning of the experiment. Then, when a ciphertext $c \leftarrow \mathsf{Encrypt}_\mathsf{PKE}(\mathsf{pk}_\mathsf{ZE}, \tau)$ is generated in the subroutine Sign of the oracle $\mathcal{O}_{\mathsf{Sign}}$, a new entry $(c, \tau, \{X_j, Y_j\}_{j\in[\![n]\!]})$

is added to $\tilde{L}$. The $\mathcal{O}^b_{\mathsf{LoRSanit}}$ oracle runs the following modified version of the subroutine $\tilde{\mathsf{Sanitize}}$: On input a certain $\tilde{c}$, it first checks whether there is an entry $(\tilde{c}, \tilde{\tau}, \{\tilde{X}_j, \tilde{Y}_j\}_{j\in[\![n]\!]})$ in $\tilde{L}$ and, if so, proceeds by setting $\tau = \tilde{\tau}$. Otherwise the algorithm aborts. In addition, when a ciphertext is generated as $c' \leftarrow \mathsf{Multiply}_{\mathsf{PKE}}(\mathsf{pk}_{\mathsf{ZE}}, c, s)$, it does the same search for the plaintext as before to get $\tilde{\tau}$ and adds a new entry $(c', \tilde{\tau} \cdot s, \{X'_j, Y'_j\}_{j\in[\![n]\!]})$ to $\tilde{L}$.

Hyb^b_2 : Defined as Hyb^b_1, except that the subroutine $\tilde{\mathsf{Sanitize}}$ is modified to replace the calling of $\mathsf{Multiply}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathbf{SKZ}_{i,j}, s)$ by searching for $\tilde{\tau}$ in $\tilde{L}$ which is the decryption of $\mathbf{SKZ}_{i,j}$ and then using $\mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, s \cdot \tilde{\tau})$ to generate the modified ciphertext $\mathbf{SKZ}'_{i,j}$.

Hyb^b_3 : Defined as Hyb^b_2, except that the subroutine $\tilde{\mathsf{Sanitize}}$ is modified to compute the signatures μ' and η' as $\mu' \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (X'_1, \ldots, X'_n))$ and $\eta' \leftarrow \mathsf{Sign}_{\mathsf{EQS}}(\mathsf{sk}^{\dagger}_{\mathsf{EQS}}, (Y'_1, \ldots, Y'_n))$, respectively.

Hyb^b_4 : Defined as Hyb^b_3, except that the subroutine $\tilde{\mathsf{Sanitize}}$ samples a fresh tuple $(Z_1, \ldots, Z_n) \leftarrow \mathbb{G}_1^n$ and sets $(X'_1, \ldots, X'_n) \coloneqq (Z_1, \ldots, Z_n)$ and $(Y'_1, \ldots, Y'_n) \coloneqq (Z_1^{\tilde{y}_1}, \ldots, Z_n^{\tilde{y}_n})^s$.

Hyb^b_5 : Defined as Hyb^b_4, except that the subroutine $\tilde{\mathsf{Sanitize}}$ samples a tuple $(w_1, \ldots, w_n) \leftarrow \mathbb{Z}_q^n$ and computes σ'_j as $\mathcal{H}(j \| m'_j)^{w_j}$ and $(Y'_1, \ldots, Y'_n)$ as $(Z_1^{w_1}, \ldots, Z_n^{w_n})$.

Observe that in the experiment Hyb^b_5, the output of the $\mathcal{O}^b_{\mathsf{LoRSanit}}$ oracle in Hyb^b_5 is completely decorrelated from the random coin b. It follows that for all PPT adversaries we have that

$$\left|\mathsf{Pr}\left[\mathsf{Hyb}^0_5 = 1\right] - \mathsf{Pr}\left[\mathsf{Hyb}^1_5 = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

We now proceed by showing the indistinguishability of each pair of hybrids.

Lemma 4. *Suppose* PKE *is correct. Then for all* PPT $\mathcal{A}$ *and* $b \in \{0,1\}$ *it holds that*

$$\left|\mathsf{Pr}\left[\mathsf{Hyb}^b_0 = 1\right] - \mathsf{Pr}\left[\mathsf{Hyb}^b_1 = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Proof (of Lemma 4). The experiments differ in the fact that in Hyb^b_1 the LoRSanit oracle aborts when queried on some $\tilde{c}$ such that no entry $(\tilde{c}, \cdot)$ is present in $\tilde{L}$. This implies that $\tilde{c}$ was not produced by the signing oracle $\mathcal{O}_{\mathsf{Sign}}$, therefore also Hyb^b_0 aborts on the same input. The indistinguishability follows by the correctness of the encryption scheme. □

Lemma 5. *Suppose* PKE *is unlinkable. Then for all* PPT $\mathcal{A}$ *and* $b \in \{0,1\}$,

$$\left|\mathsf{Pr}\left[\mathsf{Hyb}^b_1 = 1\right] - \mathsf{Pr}\left[\mathsf{Hyb}^b_2 = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Proof (of Lemma 5). We create the intermediate hybrids $\mathsf{Hyb}^b_{0,n}, \mathsf{Hyb}^b_{1,1}, \ldots, \mathsf{Hyb}^b_{1,n}, \ldots, \mathsf{Hyb}^b_{k,n}$ where each $\mathsf{Hyb}^b_{i,j}$ differs from the previous hybrid (*i.e.,* $\mathsf{Hyb}^b_{i,j-1} \mid j > 1$ or $\mathsf{Hyb}^b_{i-1,n}$ otherwise) by that the ciphertext $\mathbf{SKZ}'_{i,j}$ in the $\mathsf{Sanitize}$ algorithm is generated by $\mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, s \cdot \tilde{\tau})$ instead of using $\mathsf{Multiply}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathbf{SKZ}_{i,j}, s)$. Note that $\mathsf{Hyb}^b_{0,n}$ is identical to Hyb^b_1 and $\mathsf{Hyb}^b_{k,n}$ is identical to Hyb^b_2. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathsf{Hyb}^b_{i,j}$ and the previous hybrid $\mathsf{Hyb}^b_{i',j'}$, we can construct an algorithm $\mathcal{B}$ that can break the unlinkability of PKE as the following. $\mathcal{B}$ receives pk_c from the unlinkability challenger, it will use it as the public key for the sanitizer i. The sign and prove oracles are simulated honestly. The $\mathsf{LoRSanit}$ and sanitize oracles are simulated honestly except that when generating $\mathbf{SKZ}'_{i,j}$, $\mathcal{B}$ sends $(\mathbf{SKZ}_{i,j}, s)$ to the PKE challenger and gets c' which it will use for $\mathbf{SKZ}'_{i,j}$.

This means that the advantage of $\mathcal{A}$ to distinguish $\mathsf{Hyb}^b_{i,j}$ from $\mathsf{Hyb}^b_{i',j'}$ is equal to his advantage against the unlinkability of PKE. Thus, the advantage of $\mathcal{A}$ to distinguish Hyb^b_1 from Hyb^b_2 is equal to $n \cdot k \cdot \mathsf{Adv}^{\mathsf{Unlink}}_{\mathrm{PKE},\mathcal{A}}(\lambda)$ which is negligible. □

Lemma 6. *If* EQS *perfectly adapts signatures, then for all* PPT $\mathcal{A}$ *and* $b \in \{0,1\}$*,*

$$\left|\mathsf{Pr}\left[\mathsf{Hyb}^b_2 = 1\right] - \mathsf{Pr}\left[\mathsf{Hyb}^b_3 = 1\right]\right| = 0.$$

Proof (of Lemma 6). Trivial. □

Lemma 7. *Let* $q > 2^\lambda$*. For all generic group adversary* $\mathcal{A}$ *and* $b \in \{0,1\}$ *it holds that*

$$\left|\mathsf{Pr}\left[\mathsf{Hyb}^b_3 = 1\right] - \mathsf{Pr}\left[\mathsf{Hyb}^b_4 = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Proof (of Lemma 7). First observe that in Hyb^b_4 (with a slight notation abuse)

$$(Y'_1, \ldots, Y'_n) = (Z_1^{\tilde{y}_1}, \ldots, Z_n^{\tilde{y}_n})^s = (X'_1, \ldots, X'_n)^{s(\tilde{y}_1, \ldots, \tilde{y}_n)}$$

whereas in Hyb^b_3

$$(Y'_1, \ldots, Y'_n) = (Y_1, \ldots, Y_n)^{r \cdot s} = (X_1^{\tilde{y}_1}, \ldots, X_n^{\tilde{y}_n})^{r \cdot s} = (X'_1, \ldots, X'_n)^{s(\tilde{y}_1, \ldots, \tilde{y}_n)}.$$

Therefore, if the tuple $(X'_1, \ldots, X'_n)$ has the same distribution in both experiments the indistinguishability follows. It remains to show that

$$\begin{pmatrix} G_1, & X_1, \ldots, X_n \\ & Z_1, \ldots, Z_n \end{pmatrix} \approx \begin{pmatrix} G_1, & X_1, \ldots, X_n \\ & X_1^r, \ldots, X_n^r \end{pmatrix}$$

over the random choice of $(Z_1, \ldots, Z_n, r)$. It is easy to show that the two distributions are indistinguishable by the hardness of the decisional Diffie-Hellman problem [19]. For completeness, and as a warm-up for the proof of the next

lemma, we show that this holds in the generic group model. For ease of exposition, we denote symbolically $X_k \coloneqq G_1^{x_k}$ and $Z_k \coloneqq G_1^{z_k}$ for all $k \in [\![n]\!]$. For the left distribution we can rewrite all equations that the adversary learns as symbolic degree 1 polynomials

$$\mathfrak{x}_1 x_1 + \ldots + \mathfrak{x}_n x_n + \mathfrak{z}_1 z_1 + \ldots + \mathfrak{z}_n z_n + \mathfrak{c} = 0$$

for some coefficients $(\mathfrak{x}_1, \ldots, \mathfrak{x}_n, \mathfrak{z}_1, \ldots, \mathfrak{z}_n, \mathfrak{c})$. For the right distributions we have

$$\mathfrak{x}_1 x_1 + \ldots + \mathfrak{x}_n x_n + \mathfrak{z}_1 x_1 r + \ldots + \mathfrak{z}_n x_n r + \mathfrak{c} = 0.$$

Since $(x_1, \ldots, x_n)$ and $(z_1, \ldots, z_n)$ are uniformly chosen, by Lemma 2 all coefficients in the left distribution must be 0 with all but negligible probability. The same holds for the right distribution, since r is uniformly sampled. It follows that a generic adversary cannot learn any non-trivial relation when it is given either the left or right distribution. Therefore, both distributions look identical. □

Lemma 8. *Let $q > 2^\lambda$. For all generic group adversary $\mathcal{A}$ and $b \in \{0,1\}$ it holds that*

$$\left| \mathsf{Pr}\left[\mathsf{Hyb}_4^b = 1\right] - \mathsf{Pr}\left[\mathsf{Hyb}_5^b = 1\right] \right| \leq \mathsf{negl}(\lambda).$$

Proof (of Lemma 8). The two experiments differ in the way the variables $(Y'_1, \ldots, Y'_n)$ and $(\sigma'_1, \ldots, \sigma'_n)$ are computed. Suppose that $\mathcal{H}(i\|m_i)$ and $\mathcal{H}(i\|m'_i)$ are programmed to $G_2^{t_i^0}$ and $G_2^{t_i^1}$ respectively, where $(t_1^0, t_1^1, \ldots, t_n^0, t_n^1)$ is a randomly sampled vector in $\mathbb{Z}_q^{2n}$. The indistinguishability of the two hybrids reduces to arguing about the proximity of the following distributions

$$\begin{pmatrix} G_1^{y_1}, & \ldots, G_1^{y_n}, \\ G_1, G_1^{z_1}, & \ldots, G_1^{z_n}, \\ G_1^{sy_1z_1}, & \ldots, G_1^{sy_nz_n}, \\ G_2^{t_1^0 y_1}, & \ldots, G_2^{t_n^0 y_n}, \\ G_2^{st_1^1 y_1}, & \ldots, G_2^{st_n^1 y_n}, \\ G_2, G_2^{t_1^0}, & \ldots, G_2^{t_n^0}, \\ G_2^{t_1^1}, & \ldots, G_2^{t_n^1} \end{pmatrix} \approx \begin{pmatrix} G_1^{y_1}, & \ldots, G_1^{y_n}, \\ G_1, G_1^{z_1}, & \ldots, G_1^{z_n}, \\ G_1^{w_1z_1}, & \ldots, G_1^{w_nz_n}, \\ G_2^{t_1^0 y_1}, & \ldots, G_2^{t_n^0 y_n}, \\ G_2^{t_1^1 w_1}, & \ldots, G_2^{t_n^1 w_n}, \\ G_2, G_2^{t_1^0}, & \ldots, G_2^{t_n^0}, \\ G_2^{t_1^1}, & \ldots, G_2^{t_n^1} \end{pmatrix}$$

where the LHS corresponds to the distributions in Hyb_4^b and the RHS corresponds to the distributions in Hyb_5^b. Note that all non-trivial relations that the generic attacker can learn are restricted to certain polynomials of degree at most 2. For

illustration, we can symbolically write the relations obtained from the RHS as

$$\begin{aligned}
&\sum_{i\in[\![n]\!],j\in[\![n]\!]} \mathfrak{a}_{i,j}\left(t_i^1 w_i \cdot w_j z_j\right) && + \sum_{i\in[\![n]\!],j\in[\![n]\!]} \mathfrak{b}_{i,j}\left(t_i^1 w_i \cdot y_j\right)\\
&+ \sum_{i\in[\![n]\!],j\in[\![n]\!]} \mathfrak{c}_{i,j}\left(t_i^0 y_i \cdot w_j z_j\right) && + \sum_{i\in[\![n]\!],j\in[\![n]\!]} \mathfrak{d}_{i,j}\left(t_i^0 y_i \cdot y_j\right)\\
+ &\sum_{i\in[\![n]\!],j\in[\![n]\!],b\in\{0,1\}} \mathfrak{e}_{i,j}^b\left(t_i^b \cdot w_j z_j\right) && + \sum_{i\in[\![n]\!],j\in[\![n]\!],b\in\{0,1\}} \mathfrak{f}_{i,j}^b\left(t_i^b \cdot y_j\right)\\
&+ \sum_{i\in[\![n]\!],j\in[\![n]\!]} \mathfrak{g}_{i,j}\left(t_i^1 w_i \cdot z_j\right) && + \sum_{i\in[\![n]\!]} \mathfrak{h}_i\left(t_i^1 w_i\right)\\
&+ \sum_{i\in[\![n]\!],j\in[\![n]\!]} \mathfrak{i}_{i,j}\left(t_1^0 y_i \cdot z_j\right) && + \sum_{i\in[\![n]\!]} \mathfrak{j}_i\left(t_1^0 y_i\right)\\
+ &\sum_{i\in[\![n]\!],j\in[\![n]\!],b\in\{0,1\}} \mathfrak{k}_{i,j}^b\left(t_i^b \cdot z_j\right) && + \sum_{i\in[\![n]\!],b\in\{0,1\}} \mathfrak{l}_i^b\left(t_i^b\right)\\
&+ \sum_{i\in[\![n]\!]} \mathfrak{m}_i(y_i) && + \sum_{i\in[\![n]\!]} \mathfrak{n}_i(z_i)\\
&+ \sum_{i\in[\![n]\!]} \mathfrak{o}_i(w_i z_i) && +\mathfrak{p}\\
&= 0.
\end{aligned}$$

whereas for the LHS the equation is identical except that all occurrences of w_i and w_j are replaced with $y_i \cdot s$ and $y_j \cdot s$, respectively. Since all variables are uniformly distributed, by Lemma 2 we have that the coefficient of each unique monomial must be 0 with all but negligible probability. It is left to argue that each non-trivial relation obtained on the RHS imply also a corresponding non-trivial relation on the LHS, and vice-versa. By inspection we isolate the pairs

$$\left(\sum_{i\in[\![n]\!],j\in[\![n]\!],b\in\{0,1\}} \mathfrak{e}_{i,j}^b\left(t_i^b \cdot w_j z_j\right), \sum_{i\in[\![n]\!],j\in[\![n]\!]} \mathfrak{g}_{i,j}\left(t_i^1 w_i \cdot z_j\right)\right)$$

and

$$\left(\sum_{i\in[\![n]\!],j\in[\![n]\!],b\in\{0,1\}} \mathfrak{f}_{i,j}^b\left(t_i^b \cdot y_j\right), \sum_{i\in[\![n]\!]} \mathfrak{j}_i(t_i^0 y_i)\right)$$

that have potentially common monomials. For the latter case it is enough to observe that the monomials are identical for both the LHS and the RHS distributions as they are independent of w_i and s for all $i \in [\![n]\!]$. Therefore, if

$$\sum_{i\in[\![n]\!],j\in[\![n]\!],b\in\{0,1\}} \mathfrak{f}_{i,j}(t_i^b \cdot y_j) + \sum_{i\in[\![n]\!]} \mathfrak{j}_i(t_i^0 y_i) = 0$$

in the RHS then so it does in the LHS, and vice versa. For the former case we have that collisions occur only when $i = j$ and $b = 1$, as otherwise the monomials

are distinct and therefore any non-trivial set of coefficients will not cancel out (with very high probability). Setting $i = j$ and $b = 1$, for the RHS we have the following constraint

$$\sum_{i \in [\![n]\!]} \mathfrak{e}_{i,i}^1 \left(t_i^1 w_i z_i\right) + \sum_{i \in [\![n]\!]} \mathfrak{g}_{i,i} \left(t_i^1 w_i z_i\right) = 0$$

which implies that, with overwhelming probability, for all $i \in [\![n]\!]$ it holds that $\mathfrak{e}_{i,i}^1 = -\mathfrak{g}_{i,i}$. Applying this constraint to the LHS we obtain a corresponding non-trivial relation

$$\sum_{i \in [\![n]\!]} \mathfrak{e}_{i,i}^1 \left(st_i^1 y_i z_i\right) + \sum_{i \in [\![n]\!]} \mathfrak{g}_{i,i} \left(st_i^1 y_i z_i\right) = 0.$$

The reverse direction holds with an identical argument. Since there is a bijection between the non-trivial relations on the LHS and those on the RHS, we can conclude that the view of $\mathcal{A}$ in the two cases are indistinguishable. □

This yields the following advantage against unlinkability

$$\mathsf{Adv}_{\text{k-SAN},\mathcal{A}}^{\mathsf{Unlink},k,b}(\lambda) = \mathsf{Adv}_{\text{VRS},\mathcal{A}}^{\mathsf{Unforg},k+1}(\lambda) + n \cdot k \cdot \mathsf{Adv}_{\text{PKE},\mathcal{A}}^{\mathsf{Unlink}}(\lambda) + 2 \cdot \mathsf{Adv}_{\text{EQS},\mathcal{A}}^{\mathsf{Per-Adapt}}(\lambda) + \mathsf{Adv}_{\text{k-SAN},\mathcal{A}}^{\mathsf{Lemma\ 7}}(\lambda) + \mathsf{Adv}_{\text{k-SAN},\mathcal{A}}^{\mathsf{Lemma\ 8}}(\lambda)$$

which is negligible because n and k are small in real applications and assuming the security of the underlying primitives. □

Proof (Sanitizer Anonymity). Sanitizer anonymity relies on $\mathsf{Exp}_{\text{k-SAN},\mathcal{A}}^{\mathsf{SanitAnon},k,b}$ which can only be broken if the proof done using VRS does not maintain the anonymity of the signer within the ring and $\mathsf{Exp}_{\text{k-SAN},\mathcal{A}}^{\mathsf{AdmSec},k,b}$ which can only be broken if the adversary breaks the secrecy of the admissibility matrix $\mathbf{A}$.

If we have access to an efficient adversary $\mathcal{A}$ against $\mathsf{Exp}_{\text{k-SAN},\mathcal{A}}^{\mathsf{SanitAnon},k,b}$, we can create an algorithm $\mathcal{B}$ to break $\mathsf{Exp}_{\text{VRS},\mathcal{B}}^{\mathsf{Anon},k+1,b}$ which works as follows. $\mathcal{B}$ simulates $\mathsf{Exp}_{\text{k-SAN},\mathcal{A}}^{\mathsf{SanitAnon},k,b}$ honestly except the following. $\mathcal{B}$ receives $k+1$ public keys from the VRS challenger which it will use for its k sanitizers and signer instead of the keys generated by the $\mathsf{KGen_Z}$ algorithm. $\mathcal{A}$ outputs $(m, i_0, i_1, j', m'_{j'})$. Then, $\mathcal{B}$ makes the message part j' admissible for sanitizers i_0 and i_1 chosen by $\mathcal{A}$ and honestly generates the signature. During the sanitization, $\mathcal{B}$ will call the $\mathcal{O}_{\mathsf{LoRSign}}$ oracle of the VRS challenger on $(\mathsf{pk}_{\mathsf{ZP}}^{\mathsf{i_0}}, \mathsf{pk}_{\mathsf{ZP}}^{\mathsf{i_1}}, \mathsf{pk_S} \| \mathsf{MOD}(m) \| \mathbf{PKZ} \| \sigma_{\mathsf{SS}}', \{\mathsf{pk_{SP}}, \{\mathsf{pk}_{\mathsf{ZP}}^{\mathsf{i}}\}_{i \in [\![k]\!]}\})$ to get σ_{VRS}'. Finally, $\mathcal{B}$ sends the signature to $\mathcal{A}$ and receives its response which indicates whether i_0 or i_1 did the sanitization, *i.e.,* generated the VRS signature. $\mathcal{B}$ can use this information to respond to the anonymity challenger of VRS. Thus, the advantage against the sanitizer anonymity experiment will be

$$\mathsf{Adv}^{\mathsf{SanitAnon},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{Anon},k+1}_{\text{VRS},\mathcal{A}}(\lambda)$$

which is negligible because of the anonymity of VRS.

Regarding the experiment $\mathsf{Exp}^{\mathsf{AdmSec},k,b}_{\text{k-SAN},\mathcal{A}}$, given an efficient adversary $\mathcal{A}$ against it, we can create an algorithm $\mathcal{B}$ to break $\mathsf{Exp}^{\mathsf{IND-CPA},b}_{\text{PKE},\mathcal{B}}$ which works as follows. $\mathcal{B}$ simulates $\mathsf{Exp}^{\mathsf{AdmSec},k,b}_{\text{k-SAN},\mathcal{A}}$ honestly except for the following. $\mathcal{B}$ interacts with k IND-CPA challengers and receives a key pk_c from each of them. It will use these keys instead of the honestly generated keys for the sanitizers. $\mathcal{A}$ outputs (m, i', j'). Then, $\mathcal{B}$ will create the tuple $\tau := (0, y_{j'})$ where $y_{j'}$ is the BLS-like secret key for $m_{j'}$. It will send τ to the IND-CPA challenger of sanitizer i'. The PKE challenger returns a challenge c which will be used as $\mathbf{SKCH}_{i',j'}$ during the k-SAN signing process. Other encryption operations are done using the encryption oracle provided by the corresponding IND-CPA challenger. Then, the signature is given to $\mathcal{A}$. Finally, when $\mathcal{A}$ outputs its response indicating if the message block j' is admissible for sanitizer i', $\mathcal{B}$ can use this information to respond to the i' IND-CPA challenger. This means that the advantage against the admissibility matrix secrecy experiment will be

$$\mathsf{Adv}^{\mathsf{AdmSec},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{IND-CPA}}_{\text{PKE},\mathcal{A}}(\lambda)$$

which is negligible because of the IND-CPA security of PKE.

Thus, the total advantage of $\mathcal{A}$ against the sanitizer anonymity property is $\mathsf{Adv}^{\mathsf{Anon},k+1}_{\text{VRS},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{IND-CPA}}_{\text{PKE},\mathcal{A}}(\lambda)$ which is negligible as it is the sum of two negligible values. □

D High-Level Flow Charts

Figures 8 and 7 show how the Sign and Sanitize algorithms work on a high-level in both constructions.

E Proofs for The FSV-k-SAN Construction

Proof (Correctness). The correctness of FSV-k-SAN relies directly on the correctness of the underlying signature schemes SIG and VRS and the chameleon hash CHash. □

Proof (Immutability). There are four possible attacks against immutability, forge the signature, decrypt the chameleon hash trapdoor without having access to the secret key, find a collision to the chameleon hash without having access to the trapdoor, break the uniqueness of the randomness of CHash. We eliminate these attacks one by one in the sequence of hybrid experiments below.

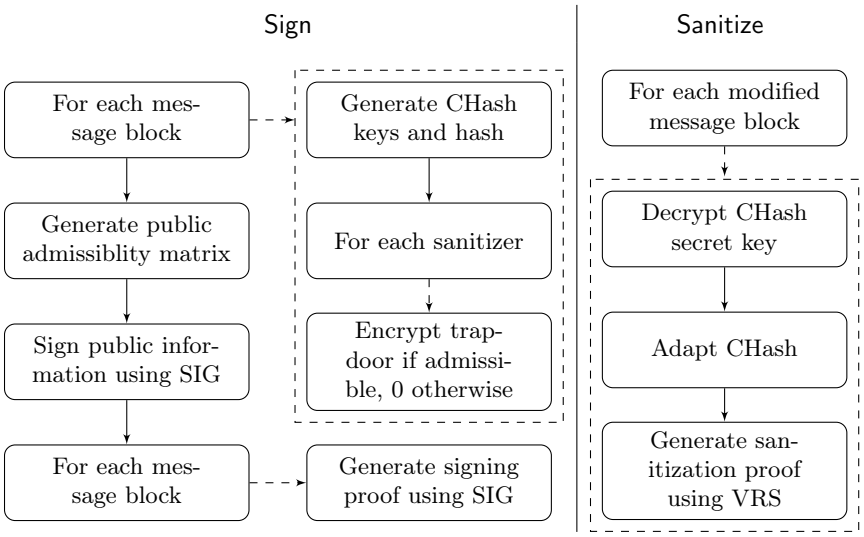

Fig. 7: High-level flow chart showing how the Sign and Sanitize algorithms work in FSV-k-SAN. A dashed line indicates that the action is done inside a loop and a dashed box groups multiple actions that are done inside the same loop.

Hyb_0 : Same as $\mathsf{Exp}^{\mathsf{Immut},k}_{\text{k-SAN},\mathcal{A}}$.

Hyb_1 : As Hyb_0, but the challenger aborts if the adversary outputs a tuple $(\mathbf{PKZ}^*, m^*, \mathbf{A}^*, \sigma^*)$ where σ^* is parsed as $(s^*, \mathbf{CH}^*, \mathbf{SKCH}^*, \mathbf{PA}^*, n^*, \rho^*)$ and s^* is a forgery on $((\mathbf{CH}^*_j.h, \mathbf{CH}^*_j.\mathsf{pk}_{\mathsf{CH}})_{j\in[\![n^*]\!]}, \mathbf{SKCH}^*, \mathbf{PA}^*, \mathsf{pk}^\dagger_{\mathsf{S}}, \mathbf{PKZ}^*, n^*)$ for SIG. We will call this abort event E_1.

Hyb_2 : As Hyb_1, but we replace all the chameleon hash trapdoors encrypted in $\mathbf{SKCH}$ as $\mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathsf{sk}^{\mathsf{j}}_{\mathsf{CH}})$ by $\mathsf{Encrypt}_{\mathsf{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, 0)$ for all sanitizers to which the adversary does not have access, *i.e.,* $(\mathsf{pk}_{\mathsf{ZE}}, \mathsf{pk}_{\mathsf{ZP}}) \in \mathbf{PKZ}^\dagger$.

Hyb_3 : As Hyb_2, but the challenger aborts if the adversary outputs a forgery $(\mathbf{PKZ}^*, m^*, \mathbf{A}^*, \sigma^*)$ where the message m^* cannot be derived from any returned signature from the sign oracle. This means that for all $(m, \mathbf{A}, \sigma)$ generated by the sign oracle of k-SAN with $\sigma.s = \sigma^*.s$, the modification from m to m^* is not admissible for the adversary's sanitizers $(\mathbf{PKZ}^* \setminus \mathbf{PKZ}^\dagger)$. This abort event is called E_2.

Hyb_4 : As Hyb_3, but the challenger aborts if the adversary outputs a valid forgery $(\mathbf{PKZ}^*, m^*, \mathbf{A}^*, \sigma^*)$ where for some $j \in [\![n]\!]$, the tuple $(\sigma^*.\mathbf{CH}_j.\mathsf{pk}_{\mathsf{CH}}, \sigma^*.\mathbf{CH}_j.h, \sigma^*.\mathbf{CH}_j.r, m^*_j)$ was not generated by the sign oracle. We call this abort event E_3.

If there exists an adversary $\mathcal{A}$ with non-negligible advantage against Hyb_1, we can construct an algorithm $\mathcal{B}$ to break the EUF-CMA security of SIG which

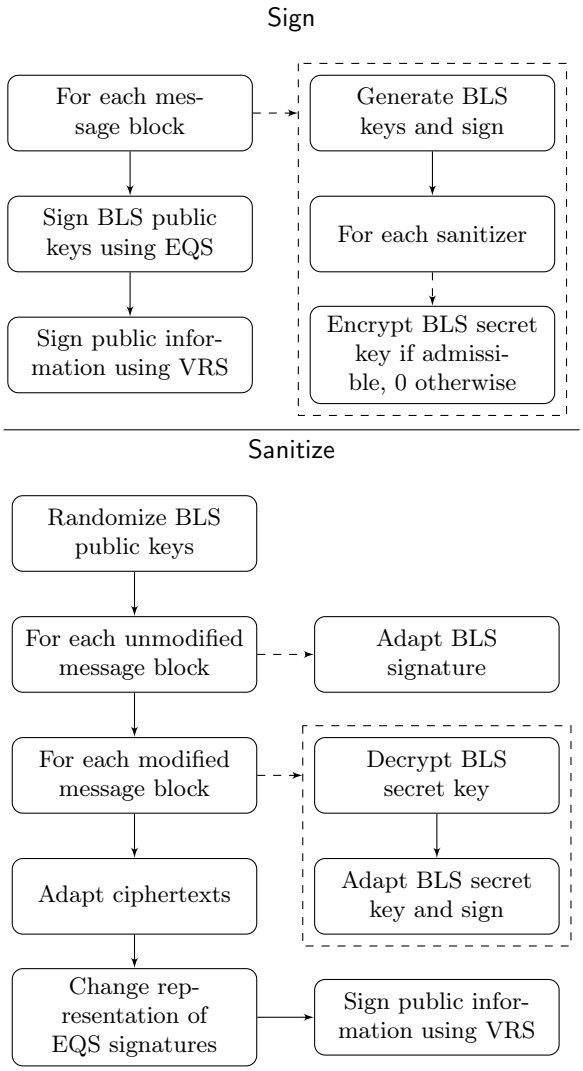

Fig. 8: High-level flow chart showing how the Sign and Sanitize algorithms work in IUT-k-SAN. A dashed line indicates that the action is done inside a loop and a dashed box groups multiple actions that are done inside the same loop.

works as the following. $\mathcal{B}$ simulates Hyb_1. $\mathcal{B}$ obtains a public key pk_c from the SIG challenger which it will use for $\mathsf{pk}_\mathsf{S}^\dagger$ instead of the key generated by the KGen_S algorithm. For every signature s or $\rho_j \in \rho$ which needs to be generated by $\mathcal{B}$ based on $\mathcal{A}$'s requests to the sign oracle of k-SAN, $\mathcal{B}$ uses the sign oracle of SIG. When E_1 happens, $\mathcal{B}$ outputs the signature that triggered E_1 and its corresponding message as a forgery to the SIG challenger. Thus, $|\mathsf{Adv}_\mathcal{A}^{\mathsf{Hyb}_1}(\lambda) - \mathsf{Adv}_\mathcal{A}^{\mathsf{Hyb}_0}(\lambda)| \leq \mathsf{Adv}_{\mathrm{SIG},\mathcal{A}}^{\mathsf{EUF-CMA}}(\lambda)$.

Note: We now know that tampering with the signature of SIG is not possible as well as impersonating the signer to generate new signatures. Furthermore, the public keys $(\mathsf{pk}_\mathsf{S}, \mathbf{PKZ})$ and n are signed as well using SIG, thus, it is impossible to add or remove sanitizers or message blocks.

We recall that the challenger of immutability generates a set of sanitizers $\mathbf{PKZ}^\dagger$ to which the adversary does not have access and the adversary can generate keys for his own sanitizers.

We create the intermediate hybrids $\mathsf{Hyb}_{0,n}, \mathsf{Hyb}_{1,1}, \ldots, \mathsf{Hyb}_{1,n}, \ldots, \mathsf{Hyb}_{k,n}$ where each $\mathsf{Hyb}_{i,j}$ differs from the previous hybrid (*i.e.,* $\mathsf{Hyb}_{i,j-1} \mid j > 1$ or $\mathsf{Hyb}_{i-1,n}$ otherwise) by that the ciphertext $\mathbf{SKCH}_{i,j}$ is an encryption of 0 instead of $\mathsf{sk}_{\mathsf{CH}}^{\mathsf{j}}$ if the adversary does not have access to the sanitizer i, *i.e.,* $(\mathsf{pk}_{\mathsf{ZE}}^{\mathsf{i}}, \mathsf{pk}_{\mathsf{ZP}}^{\mathsf{i}}) \in \mathbf{PKZ}^\dagger$. Note that $\mathsf{Hyb}_{0,n}$ is identical to Hyb_1 and $\mathsf{Hyb}_{k,n}$ is identical to Hyb_2. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathsf{Hyb}_{i,j}$ and the previous hybrid $\mathsf{Hyb}_{i',j'}$, we can construct an algorithm $\mathcal{B}$ that can break the IND-CPA security of PKE as the following. $\mathcal{B}$ receives pk_c from the IND-CPA challenger, it will use it in the public key for the sanitizer i instead of the one generated by KGen_Z if $(\mathsf{pk}_{\mathsf{ZE}}^{\mathsf{i}}, \mathsf{pk}_{\mathsf{ZP}}^{\mathsf{i}}) \in \mathbf{PKZ}^\dagger$. The sign oracle is simulated honestly except for the ciphertexts for the sanitizer i if $(\mathsf{pk}_{\mathsf{ZE}}^{\mathsf{i}}, \mathsf{pk}_{\mathsf{ZP}}^{\mathsf{i}}) \in \mathbf{PKZ}^\dagger$. For $\mathbf{SKCH}i, j$, $\mathcal{B}$ defines two messages τ_0 and τ_1 as

$$\tau_0 = \begin{cases} \mathsf{sk}_{\mathsf{CH}}^{\mathsf{j}}, & a_{i,j} = 1 \\ 0, & \text{otherwise} \end{cases} \quad \tau_1 = \begin{cases} \mathsf{sk}_{\mathsf{CH}}^{\mathsf{j}}, & a_{i,j} = 1 \wedge (\mathsf{pk}_{\mathsf{ZE}}^{\mathsf{i}}, \mathsf{pk}_{\mathsf{ZP}}^{\mathsf{i}}) \notin \mathbf{PKZ}^\dagger \\ 0, & \text{otherwise} \end{cases}.$$

Then, $\mathcal{B}$ sends (τ_0, τ_1) to the IND-CPA challenger and gets the ciphertext c which it will use for $\mathbf{SKCH}_{i,j}$. For the other values in $\mathbf{SKCH}_i$, $\mathcal{B}$ will use the encryption oracle of the IND-CPA challenger.

When $\mathcal{A}$ produces his response b^*, $\mathcal{B}$ can send b^* as his response to the IND-CPA challenger. This means that the advantage of $\mathcal{A}$ to distinguish $\mathsf{Hyb}_{i,j}$ from $\mathsf{Hyb}_{i',j'}$ is equal to his advantage against the IND-CPA of PKE. Thus, the advantage of $\mathcal{A}$ to distinguish Hyb_1 from Hyb_2 is equal to $n \cdot k \cdot \mathsf{Adv}_{\mathrm{PKE},\mathcal{A}}^{\mathsf{IND-CPA}}(\lambda)$ as we have $n \cdot k$ ciphertexts. As IND-CPA implies one-wayness, then we know that the adversary cannot break the encryption for sanitizers to which he does not have access which gives us $|\mathsf{Adv}_\mathcal{A}^{\mathsf{Hyb}_2}(\lambda) - \mathsf{Adv}_\mathcal{A}^{\mathsf{Hyb}_1}(\lambda)| \leq n \cdot k \cdot \mathsf{Adv}_{\mathrm{PKE},\mathcal{A}}^{\mathsf{IND-CPA}}(\lambda)$.

If there exists an adversary $\mathcal{A}$ with non-negligible advantage against Hyb_3, we can construct an algorithm $\mathcal{B}$ to break the collision resistance of CHash which

works as the following. First, let's recall that the signatures generated by the sign oracle are kept in Σ. Now we move to the construction of $\mathcal{B}$ which will simulate Hyb_3. During the signing process, $\mathcal{B}$ will start a collision resistence challenge for each chameleon hash of an inadmissible block to $\mathbf{PKZ} \setminus \mathbf{PKZ}^{\dagger}$. It will replace $\mathsf{pk}_{\mathsf{CH}}$ by pk_c - which it received from the CHash challenger - and use the hash oracle of CHash to calculate the hash values. Let the signature returned by $\mathcal{A}$ be σ^* which is parsed as $(s^*, \mathbf{CH}^*, \mathbf{SKCH}^*, \mathbf{PA}^*, n^*)$. As we have established from the previous hybrids, $\mathcal{A}$ cannot tamper with the signature nor decrypt the trapdoors encrypted for sanitizers in $\mathbf{PKZ}^{\dagger}$. Thus, we must have a signature $(\sigma', m') \in \Sigma$ which have the same values as σ except for $\sigma'.\mathbf{CH}$, and an index $j \in [\![n]\!]$ for which we have

$$\begin{aligned}\mathsf{Check}_{\mathsf{CHash}}(\mathsf{pk}_{\mathsf{CH}}, (j\|m_j^*), \mathbf{CH}_j^*.r, \mathbf{CH}_j^*.h) = \\ \mathsf{Check}_{\mathsf{CHash}}(\mathsf{pk}_{\mathsf{CH}}, (j\|m_j'), \sigma'.\mathbf{CH}_j.r, \mathbf{CH}_j^*.h) = 1\end{aligned}$$

where $m_j^* \neq m_j'$. Thus, $\mathcal{B}$ can output the tuple $((j\|m_j^*), \mathbf{CH}_j^*.r, (j\|m_j'), \sigma'.\mathbf{CH}_j.r, \mathbf{CH}_j^*.h)$ as a collision to the corresponding CHash challenger. As we have n hashes, $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_3}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_2}(\lambda)| \leq n \cdot \mathsf{Adv}_{\mathrm{CHash},\mathcal{A}}^{\mathsf{Col-Res}}(\lambda)$.

If there exists an adversary $\mathcal{A}$ that can trigger the abort event E_3 in Hyb_4 with non-negligible probability, we can create an efficient algorithm $\mathcal{B}$ against the uniqueness of CHash which works as the following. $\mathcal{B}$ simulates Hyb_4 honestly except that $\mathcal{B}$ will interact with the uniqueness challenger of CHash during the setup phase to get $\mathsf{pp}_{\mathsf{CH}}$. Then, when E_3 is triggered, this means that the adversary produced a forgery $(\mathbf{PKZ}^*, m^*, \mathbf{A}^*, \sigma^*)$ where for some $j \in [\![n]\!]$, the tuple $(\sigma^*.\mathbf{CH}_j.\mathsf{pk}_{\mathsf{CH}}, \sigma^*.\mathbf{CH}_j.h, \sigma^*.\mathbf{CH}_j.r, m_j^*)$ does not exist in the history and thus not generated by the algorithm $\mathsf{Hash}_{\mathsf{CHash}}$ in the sign oracle. As we have shown in previous hybrids, the adversary cannot tamper with signed values using SIG like $\sigma^*.\mathbf{CH}_j.\mathsf{pk}_{\mathsf{CH}}$ and $\sigma^*.\mathbf{CH}_j.h$, and that CHash is collision resistant, this means that there must exist a tuple $(\mathsf{pk}_{\mathsf{CH}}', h', r', m')$ in the history such that $\mathsf{pk}_{\mathsf{CH}}' = \sigma^*.\mathbf{CH}_j.\mathsf{pk}_{\mathsf{CH}} \wedge h' = \sigma^*.\mathbf{CH}_j.h \wedge m_j^* = m'$ but with $r' \neq \sigma^*.\mathbf{CH}_j.r$. Since the forgery is valid, this means that

$$\begin{aligned}\mathsf{Check}_{\mathsf{CHash}}(\mathsf{pk}_{\mathsf{CH}}', (j\|m'), r', h') = \\ \mathsf{Check}_{\mathsf{CHash}}(\mathsf{pk}_{\mathsf{CH}}', (j\|m'), \sigma^*.\mathbf{CH}_j.r, h') = 1.\end{aligned}$$

Thus, $\mathcal{B}$ can return $(\mathsf{pk}_{\mathsf{CH}}', h', r', r^*, m')$ as a forgery to the uniqueness challenger of CHash. As we have n hashes, $|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_4}(\lambda) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hyb}_3}(\lambda)| \leq n \cdot \mathsf{Adv}_{\mathrm{CHash},\mathcal{A}}^{\mathsf{Uniq}}(\lambda)$.

In summary, we have shown that the adversary cannot forge the SIG signature because of its EUF-CMA security, cannot break the encryption of trapdoors because of the IND-CPA security of PKE, cannot find collisions for the chameleon hashes because of their collision resistance, and the randomness of the chameleon hash is unique. Thus, we have the following advantage against the immutability

game

$$\mathsf{Adv}^{\mathsf{Immut},k}_{\text{k-SAN},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{EUF-CMA}}_{\text{SIG},\mathcal{A}}(\lambda) + n \cdot k \cdot \mathsf{Adv}^{\mathsf{IND-CPA}}_{\text{PKE},\mathcal{A}}(\lambda) + n \cdot \mathsf{Adv}^{\mathsf{Col-Res}}_{\text{CHash},\mathcal{A}}(\lambda) + n \cdot \mathsf{Adv}^{\mathsf{Uniq}}_{\text{CHash},\mathcal{A}}(\lambda)$$

which is negligible because of the security of SIG, PKE, and CHash. Moreover, the values of n and k are small in real applications (around 10) which means that multiplying the already negligible values by them does not make the full advantage non-negligible. □

Proof (Signer-Accountability). Let us consider that there exists an efficient adversary $\mathcal{A}$ against $\mathsf{Exp}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}$. We can use this adversary to create an algorithm $\mathcal{B}$ to break $\mathsf{Exp}^{\mathsf{Unforg},k}_{\text{VRS},\mathcal{B}}$ as the following. $\mathcal{B}$ simulates $\mathsf{Exp}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}$ honestly except for the following. It receives k public keys from the VRS challenger, then, it will use these keys for its k sanitizers instead of the ones generated in KGen_Z. $\mathcal{B}$ will use the sign oracle of VRS to generate any required VRS signatures inside the sanitization oracle of k-SAN. $\mathcal{A}$ eventually outputs the forgery $(\mathsf{pk}^*_\mathsf{S}, m^*, \sigma^*)$ which satisfies the conditions set in $\mathsf{Exp}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}$, *i.e.,* $\mathsf{Judge}(\mathsf{pk}^*_\mathsf{S}, \mathbf{PKZ}^\dagger, m^*, \sigma^*, \pi^*, \perp) = Z$, $\mathsf{Verify}(\mathsf{pk}^*_\mathsf{S}, \mathbf{PKZ}^\dagger, m^*, \sigma^*) = 1$, and $(\mathsf{pk}^*_\mathsf{S}, m^*, \sigma^*) \notin \Sigma$. Judge only returns Z when there is an admissible block according to **PA** and Verify only returns 1 if for each admissible block i according to **PA**, ρ_i is a valid VRS signature under the ring $\{\mathsf{pk}^{\dagger,\mathsf{i}}_{\mathsf{ZP}}\}_{i\in[\![k]\!]}$. Since, $(\mathsf{pk}^*_\mathsf{S}, m^*, \sigma^*) \notin \Sigma$, this means that there exists a ρ_j which was not generated using the $\mathcal{O}_{\mathsf{Sanit}}$ oracle provided by $\mathcal{B}$ which means that the signature was not generated by the sign oracle of VRS. Thus, we can return $(\{\mathsf{pk}^{\dagger,\mathsf{i}}_{\mathsf{ZP}}\}_{i\in[\![k]\!]}, (j\|m^*_j\|s^*), \sigma^*.\rho_j)$ as the response to the VRS challenger. As we have at most n VRS signatures, the advantage against singer accountability will be

$$\mathsf{Adv}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}(\lambda) \leq n \cdot \mathsf{Adv}^{\mathsf{Unforg},k}_{\text{VRS},\mathcal{A}}(\lambda)$$

which is negligible because of the unforgeability of VRS and because n is small in real applications. □

Proof (Sanitizer-Accountability). If there exists an adversary $\mathcal{A}$ that has a non-negligible advantage against $\mathsf{Exp}^{\mathsf{SanAcc}}_{\text{k-SAN},\mathcal{A}}$, then we can create an algorithm $\mathcal{B}$ that can break the EUF-CMA security of SIG as the following. $\mathcal{B}$ simulates $\mathsf{Exp}^{\mathsf{SanAcc}}_{\text{k-SAN},\mathcal{A}}$ except for the following. $\mathcal{B}$ gets a public key pk_c from the SIG challenger which it will use for $\mathsf{pk}^\dagger_\mathsf{S}$ instead of the key generated by the KGen_S algorithm. For every signature s or $\rho_j \in \rho$ which needs to be generated by $\mathcal{B}$ based on $\mathcal{A}$'s requests to the sign oracle of k-SAN, $\mathcal{B}$ uses the sign oracle of SIG. The prove oracle is simulated honestly. If $\mathcal{A}$ succeeds, then he has produced a signature $(\mathbf{PKZ}^*, m^*, \sigma^*)$ such that $\mathsf{Verify}(\mathsf{pk}^\dagger_\mathsf{S}, \mathbf{PKZ}^*, m^*, \sigma^*) = 1$, $(\mathbf{PKZ}^*, m^*, \sigma^*) \notin \Sigma$, and $\mathsf{Judge}(\mathsf{pk}^\dagger_\mathsf{S}, \mathbf{PKZ}^*, m^*, \sigma^*, \pi^*, \perp) \neq Z$. In FSV-k-SAN, Judge returns S on a signature σ^* with $j = \perp$ if and only if there are no admissible blocks in the

message, *i.e.,* $\nexists j \in [\![n]\!], \sigma^*.\mathbf{PA}_j = 1$. In the Verify algorithm, we note that in order for it to return 1, the signature $\sigma^*.s$ should be valid and for each message block j where $\sigma^*.\mathbf{PA}_j = 0$ we should have $\sigma^*.\rho_j$ which is a valid SIG signature on $(j \| m_j^* \| \sigma^*.s)$. As $(\mathbf{PKZ}^*, m^*, \sigma^*) \notin \Sigma$, this means that at least $\sigma^*.s$ or a $\sigma^*.rho_j$ for some $j \in [\![n]\!]$ was not generated by the sign oracle of SIG. Thus, $\mathcal{B}$ can return that signature as a forgery to the SIG challenger. As we have at most $n+1$ signatures generated using SIG, we have

$$\mathsf{Adv}^{\mathsf{SanAcc}}_{\text{k-SAN},\mathcal{A}}(\lambda) \leq (n+1) \cdot \mathsf{Adv}^{\mathsf{EUF-CMA}}_{\text{SIG},\mathcal{A}}(\lambda)$$

which is negligible because of the EUF-CMA security of SIG and because n is small in real applications. □

Proof (Privacy). In the Sign algorithm, we include two information related to the message in the signature, the CHash hash value and randomness and the SIG signature proof in ρ. The signature proof is replaced by a new one using VRS to prove sanitization, so it cannot be used to attack privacy. Regarding the hash value, we will define a series of hybrid experiments $\mathsf{Hyb}_0, \mathsf{Hyb}_1, \ldots, \mathsf{Hyb}_n$ where Hyb_0 is the same as $\mathsf{Exp}^{\mathsf{Inv},k,b}_{\text{k-SAN},\mathcal{A}}$ and each consecutive experiment Hyb_j differs from the previous one Hyb_{j-1} by that the Sanitize algorithm is modified as the following. The algorithm will generate a new hash using $\mathsf{Hash}_{\mathsf{CHash}}$ instead of using $\mathsf{Adapt}_{\mathsf{CHash}}$ for the message block j, and generate a new SIG signature using the new hash value. If an adversary $\mathcal{A}$ can distinguish between any two consecutive experiments Hyb_j and Hyb_{j-1}, then we can use it to construct an algorithm $\mathcal{B}$ that can break the indistinguishability of CHash as the following. $\mathcal{B}$ will simulate Hyb_j honestly except for the following. $\mathcal{B}$ will receive pk_c from the CHash challenger which it will use for $\mathsf{pk}_{\mathsf{CH}}$ for the message block j instead of the one generated by the $\mathsf{KGen}_{\mathsf{CHash}}$ algorithm in Sign. In the Sanitize algorithm, $\mathcal{B}$ will use the $\mathcal{O}^b_{\mathsf{HashOrAdapt}}$ oracle of the CHash challenger to generate the hash value and the randomness. Depending on the value of b which was chosen by the CHash challenger, $\mathcal{B}$ simulates either Hyb_j or Hyb_{j-1}. When $\mathcal{A}$ outputs his response, $\mathcal{B}$ can use it to respond to the CHash challenger. Thus, the advantage of $\mathcal{A}$ to distinguish any Hyb_j from Hyb_{j-1} is equal to his advantage against the indistinguishability of CHash. This yields

$$\mathsf{Adv}^{\mathsf{Priv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) \leq n \cdot \mathsf{Adv}^{\mathsf{Ind},b}_{\text{CHash},\mathcal{A}}(\lambda)$$

which is negligible because of the indistinguishability of CHash and because n is small in real applications. □

Proof (Full-Sanitization Verifiability). Let us consider that there exists an efficient adversary $\mathcal{A}$ against $\mathsf{Exp}^{\mathsf{FullSanit},k}_{\text{k-SAN},\mathcal{A}}$. We can use this adversary to create an algorithm $\mathcal{B}$ to break $\mathsf{Exp}^{\mathsf{Unforg},k}_{\text{VRS},\mathcal{B}}$ as the following. $\mathcal{B}$ simulates $\mathsf{Exp}^{\mathsf{FullSanit},k}_{\text{k-SAN},\mathcal{A}}$ honestly except for the following. It receives k public keys from the VRS challenger, then, it will use these keys for its k sanitizers instead of the ones generated by the $\mathsf{KGen}_{\mathsf{Z}}$ algorithm. $\mathcal{B}$ will use the sign oracle of VRS to generate any required

VRS signatures inside the sanitization oracle of k-SAN. $\mathcal{A}$ eventually outputs his response which means that $\mathsf{Verify}(\mathsf{pk}_\mathsf{S}^*, \mathbf{PKZ}^\dagger, m^*, \sigma^*) = 1$, there exists an admissible block according to $\sigma^*.\mathbf{PA}$, and $(\mathsf{pk}_\mathsf{S}^*, m^*, \sigma^*) \notin \Sigma$. Verify only returns 1 if for each admissible block j (*i.e.,* $\sigma^*.\mathbf{PA}_j = 1$), ρ_j is a valid VRS signature under the ring $\{\mathsf{pk}_{\mathsf{ZP}}^{\dagger,\mathsf{i}}\}_{i\in[\![k]\!]}$. Since, $(\mathsf{pk}_\mathsf{S}^*, m^*, \sigma^*) \notin \Sigma$, this means that there exists a ρ_j which was not generated using the $\mathcal{O}_{\mathsf{Sanit}}$ oracle provided by $\mathcal{B}$ which means that the ρ_j was not generated by the sign oracle of VRS. Thus, we can return $(\{\mathsf{pk}_{\mathsf{ZP}}^{\dagger,\mathsf{i}}\}_{i\in[\![k]\!]}, (j\|m_j^*\|s^*), \sigma^*.\rho_j)$ as the response to the VRS challenger. As we have at most n VRS signatures, the advantage against full-sanitization verifiability will be

$$\mathsf{Adv}^{\mathsf{FullSanit},k}_{\text{k-SAN},\mathcal{A}}(\lambda) \leq n \cdot \mathsf{Adv}^{\mathsf{Unforg},k}_{\text{VRS},\mathcal{A}}(\lambda)$$

which is negligible because of the unforgeability of VRS and because n is small in real applications. □

Proof (Sanitizer Anonymity).

Sanitizer anonymity relies on $\mathsf{Exp}^{\mathsf{SanitAnon},k,b}_{\text{k-SAN},\mathcal{A}}$ which can only be broken if the proof done using VRS does not maintain the anonymity of the signer within the ring and $\mathsf{Exp}^{\mathsf{AdmSec},k,b}_{\text{k-SAN},\mathcal{A}}$ which can only be broken if the adversary breaks the secrecy of the admissibility matrix $\mathbf{A}$.

If we have access to an efficient adversary $\mathcal{A}$ against $\mathsf{Exp}^{\mathsf{SanitAnon},k,b}_{\text{k-SAN},\mathcal{A}}$, we can create an algorithm $\mathcal{B}$ to break $\mathsf{Exp}^{\mathsf{Anon},k,b}_{\text{VRS},\mathcal{B}}$ which works as follows. $\mathcal{B}$ simulates $\mathsf{Exp}^{\mathsf{SanitAnon},k,b}_{\text{k-SAN},\mathcal{A}}$ honestly except for the following. First, $\mathcal{B}$ receives k public keys from the VRS challenger which it will use for its k sanitizers instead of the keys generated by the KGen_Z algorithm. $\mathcal{A}$ outputs $(m, i_0, i_1, j', m'_{j'})$. Then, $\mathcal{B}$ makes the message part j' admissible for sanitizers i_0 and i_1 chosen by $\mathcal{A}$ and honestly generates the signature. During the sanitization, $\mathcal{B}$ will call the $\mathcal{O}_{\mathsf{LoRSign}}$ oracle of the VRS challenger on $(\mathsf{pk}_{\mathsf{ZP}}^{\mathsf{i}_0}, \mathsf{pk}_{\mathsf{ZP}}^{\mathsf{i}_1}, (j'\|m_{j'}\|\sigma.s), \{\mathsf{pk}_{\mathsf{ZP}}^{\mathsf{i}}\}_{i\in[\![k]\!]})$ to generate the proof of sanitization for $m_{j'}$. Finally, $\mathcal{B}$ sends the signature to $\mathcal{A}$ and receives its response which indicates whether i_0 or i_1 did the sanitization, *i.e.,* generated the VRS signature. $\mathcal{B}$ can use this information to respond to the anonymity challenger of VRS. Thus, the advantage against the sanitizer anonymity experiment will be

$$\mathsf{Adv}^{\mathsf{SanitAnon},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{Anon},k}_{\text{VRS},\mathcal{A}}(\lambda)$$

which is negligible because of the anonymity of VRS.

Regarding the experiment $\mathsf{Exp}^{\mathsf{AdmSec},k,b}_{\text{k-SAN},\mathcal{A}}$, given an efficient adversary $\mathcal{A}$ against it, we can create an algorithm $\mathcal{B}$ to break $\mathsf{Exp}^{\mathsf{IND-CPA},b}_{\text{PKE},\mathcal{B}}$ which works as follows. $\mathcal{B}$ simulates $\mathsf{Exp}^{\mathsf{AdmSec},k,b}_{\text{k-SAN},\mathcal{A}}$ honestly except for the following. $\mathcal{B}$ interacts with k IND-CPA challengers and receives a key pk_c from each of them. It will use these keys instead of the honestly generated keys for the sanitizers. $\mathcal{A}$ outputs (m, i', j').

Then, $\mathcal{B}$ will create the tuple $\tau := (0, \mathsf{sk}_{\mathsf{CH}}^{j'})$ where $\mathsf{sk}_{\mathsf{CH}}^{j'}$ is the trapdoor of the chameleon hash for $m_{j'}$. It will send τ to the IND-CPA challenger of sanitizer i'. The PKE challenger returns a challenge c which will be used as $\mathbf{SKCH}_{i',j'}$ during the k-SAN signing process. Other encryption operations are done using the encryption oracle provided by the corresponding IND-CPA challenger. Then, the signature is given to $\mathcal{A}$. Finally, when $\mathcal{A}$ outputs its response indicating if the message block j' is admissible for sanitizer i', $\mathcal{B}$ can use this information to respond to the i' IND-CPA challenger. This means that the advantage against the admissibility matrix secrecy experiment will be

$$\mathsf{Adv}^{\mathsf{AdmSec},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{IND-CPA}}_{\text{PKE},\mathcal{A}}(\lambda)$$

which is negligible because of the IND-CPA security of PKE.

Thus, the total advantage of $\mathcal{A}$ against the sanitizer anonymity property is $\mathsf{Adv}^{\mathsf{Anon},k}_{\text{VRS},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{IND-CPA}}_{\text{PKE},\mathcal{A}}(\lambda)$ which is negligible as it is the sum of two negligible values. □

F Comparing Real and Theoretical Performance of The Implementation

We report in Table 2 the theoretical evaluation of our constructions' efficiency. The signature, proof, and key sizes are evaluated in terms of the number of group elements, while the performance is given in terms of the number of exponentiations in each group and the number of pairing operations. We note that the signer keys in FSV-k-SAN are smaller than IUT-k-SAN, but the signature can be larger depending on the number of admissible blocks. Moreover, Sign, Sanitize, and Judge are faster in FSV-k-SAN whereas Verify can be slower depending on the number of admissible blocks. We also note that the signature size and execution time of most algorithms in both constructions has a linear relation with the number of blocks n and sanitizers k. This is not a major concern for most real applications as they will have a small number of blocks and sanitizers. One of the major bottlenecks that is affected by the number of blocks and sanitizers is the ciphertexts matrix which contains nk elements. Special types of encryption schemes could be used to reduce its size, but this depends heavily on the application. For instance, broadcast encryption can be used but at the cost of invisibility and sanitizer anonymity. Another example is Attribute-Based Encryption (ABE) which is used in [18, 30]. However, as most ABE systems have a ciphertext size that is linear in the number of attributes, we will only see significant improvement if the number of attributes is significantly smaller than the number of sanitizers. Another option is to use constant-size systems such as [3] which come with different trade-offs such as restrictions on the policies that can be defined and/or linear or quadratic secret key sizes.

We now compare the implementation's performance to the theoretical complexity calculation.

Table 2: On top, the size of the signer keys $(\mathsf{sk_S}, \mathsf{pk_S})$, sanitizer keys $(\mathsf{sk_Z}, \mathsf{pk_Z})$, signature σ, and proof π. On the bottom, the dominating operations (exponentiation and pairing) in k-SAN algorithms per construction. The fields $\mathbb{Z}_{N^2}$ and $\mathbb{Z}_N$ are from the Paillier Cryptosystem. We denote by $\mathbb{Z}_q$ the prime fields of all building blocks assuming that we use similar size prime numbers for all building blocks in the implementation. In IUT-k-SAN, $\mathbb{G}_1$ and $\mathbb{G}_2$ are from the BLS12-381 curve. In FSV-k-SAN, $\mathbb{G}_1$ is from the secp256k1 curve. Also, n_0 (*resp.* n_1) denotes the number of inadmissible (*resp.* admissible) blocks according to **PA**.

Const.	$\mathsf{sk_S}$	$\mathsf{sk_Z}$	$\mathsf{pk_S}$	$\mathsf{pk_Z}$	σ	π
IUT-k-SAN	$(n+2)\ \mathbb{Z}_q$	$1\ \mathbb{Z}_q+$ $2\ \mathbb{Z}_N$	$1\ \mathbb{Z}_q+$ $(n+1)\ \mathbb{G}_2$	$1\ \mathbb{Z}_q+1\ \mathbb{Z}_N+$ $1\ \mathbb{Z}_{N^2}$	$(4k+6)\ \mathbb{Z}_q+(kn+k)\ \mathbb{Z}_{N^2}+$ $(2n+6)\ \mathbb{G}_1+(n+3)\ \mathbb{G}_2$	$5\ \mathbb{Z}_q$
FSV-k-SAN	$1\ \mathbb{Z}_q$	$1\ \mathbb{Z}_q+$ $2\ \mathbb{Z}_N$	$1\ \mathbb{G}_1$	$1\ \mathbb{Z}_q+1\ \mathbb{Z}_N+$ $1\ \mathbb{Z}_{N^2}$	$(4kn_1+3n+2n_1)\ \mathbb{Z}_q+$ $(2n_0+2)\ \mathbb{Z}_q+$ $kn\ \mathbb{Z}_{N^2}+n\ \{0,1\}$	-

Const.	Sign	Sanitize	Verify	Prove	Judge
IUT-k-SAN	$(4k+3)\ \mathbb{Z}_q+$ $(kn+k)\ \mathbb{Z}_{N^2}+$ $(4n+8)\ \mathbb{G}_1+(n+3)\ \mathbb{G}_2$	$(4k+3)\ \mathbb{Z}_q+$ $(2n+6)\ \mathbb{G}_1+(n+3)\ \mathbb{G}_2+$ $(kn+k+\lvert\mathsf{MOD}\rvert)\ \mathbb{Z}_{N^2}$	$(4k+4)\ \mathbb{G}_1$ $(4n+10)$ Pair.	$3\ \mathbb{Z}_q$	$4\ \mathbb{Z}_q$
FSV-k-SAN	$3n\ \mathbb{Z}_q+kn\ \mathbb{Z}_{N^2}+$ $(n+1)\ \mathbb{G}_1$	$\lvert\mathsf{MOD}\rvert(4k-1)\ \mathbb{Z}_q+$ $\lvert\mathsf{MOD}\rvert\ \mathbb{Z}_{N^2}$	$(4kn_1+2n)\ \mathbb{Z}_q+$ $(2n_0+2)\ \mathbb{G}_1$	-	-

To have a more accurate comparison in the test, we use similar size fields and groups in all building blocks even if some of them become insecure. This was needed because in order to have a secure implementation, we need to set $\lambda \geq 2048$ for PKE, CHash, and VRS which makes the contribution of the other building blocks negligible as the fields and groups in the used elliptic curves are much smaller.

We tested the performance of the constructions by running each algorithm 200 times given different values of k and n and calculating the average execution time. The test was executed on a Linux server with an Intel i5-11500 processor and 32GB of RAM. The number of admissible blocks was fixed to 3 and the number of modified blocks was fixed to 1 in each call to the Sanitize algorithm as these values impact the performance as explained in Section 7.

To do the comparison between theoretical and real execution time, we also need to calculate the estimated execution time based on the formulas in Table 2. We calculated the time of exponentiation in the different groups and fields and that of pairing by executing the operations 200 times on random values and calculating the average. Exponentiation in $\mathbb{Z}_q$, $\mathbb{G}_1$ for BLS12-381, $\mathbb{G}_2$ for BLS12-381, $\mathbb{G}_1$ for secp256k1, and $\mathbb{Z}_{N^2}$ took 43, 162, 458, 91, and 590 µs respectively, whereas pairing in the BLS12-381 curve took 1085 µs.

Figures 9 and 10 show the test results. In FSV-k-SAN, as k grows, the execution time of all algorithms except Judge grows, on the other hand, n only impacts the execution time of the Sign and Verify algorithms. Regarding IUT-k-SAN, the in-

crease of k and n impacts all algorithms except Judge and Prove. We see similar trends in both real and theoretical execution time in both constructions. When we first generated the curves we noticed a lot of differences between the theoretical analysis and the real execution time. After investigation, we found that the implementation in some Rust crates is significantly more efficient than the theoretical cost. For example, the Paillier cryptosystem does one exponentiation in $\mathbb{Z}_{N^2}$ for encryption, decryption, and scalar multiplication. But the real execution time shows that encryption is slightly slower than decryption which is itself slower than scalar multiplication. Also, exponentiation in the kzen-paillier crate is 3 times more efficient than the one in num_bigint. Moreover, the implementation of the verification of Mercurial signatures in the delegatable_credentials crate is much more efficient than the theoretical calculation. Adjusting for these factors helps in getting a more accurate comparison. We applied the following adjustments to the theoretical analysis: the encryption, decryption, and scalar multiplication in the Paillier cryptosystem take 156, 138, and 90 µs respectively, and the verification in Mercurial signatures takes approximately $46n + 2876$ µs.

We did the same test on a Google Pixel 6a smartphone. We see that the execution time nearly doubled compared to the server which is expected given that we passed to a less powerful device. In particular, the Verify algorithm seem to be more affected than other algorithms in both constructions. We investigated this and found that for FSV-k-SAN, exponentiation in $\mathbb{Z}_q$ increased from 43 µs on the server to 91 µs on the smartphone whereas exponentiation in $\mathbb{G}_1$ for the secp256k1 curve increased from 91 µs to 128 µs. This disproportionate increase in the cost of operations within $\mathbb{Z}_q$ results in a comparatively greater contribution of $\mathbb{Z}_q$ exponentiation to the overall execution time on the smartphone than on the server. For IUT-k-SAN, the delegatable_credentials crate seem to suffer from approximately a 10 times slowdown in the verification algorithm when executing it on the smartphone. We compared its performance to the mercurial-signature crate and found that it performs better than the delegatable_credentials crate on the smartphone but worse on the server. Precisely, the verification of a signature on a 15 elements message in the delegatable_credentials crate took ∼4 ms on the server and ∼43 ms on the smartphone, while the mercurial-signature crate took ∼20 ms on the server and ∼30 ms on the smartphone. A major difference between the two crates is that the delegatable_credentials crate uses the multi Miller loop optimization [32] whereas the mercurial-signature crate calculates the pairings one by one. We maintained the use of the delegatable_credentials crate as the gain of performance on the server side is more significant than the loss of performance on the smartphone side. The results can be found in Figures 11 and 12.

G Efficiency Comparison With the Existing Literature

We compare the efficiency of our constructions to the existing multi-sanitizer schemes [1, 7, 14, 15, 25, 30]. Only [1] calculated the size and complexity of their construction in terms of number of group elements and operations. Other papers

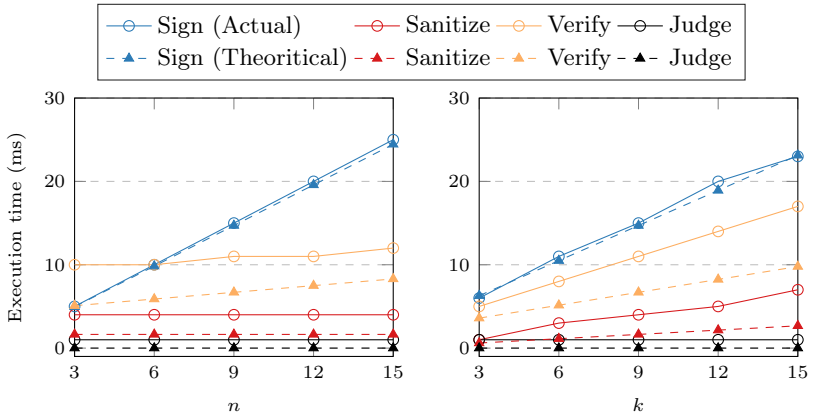

Fig. 9: Theoretical and real performance of FSV-k-SAN on the server. In the first figure, we fix $k = 9$ and test with variable n. In the second one, we fix $n = 9$ and test k.

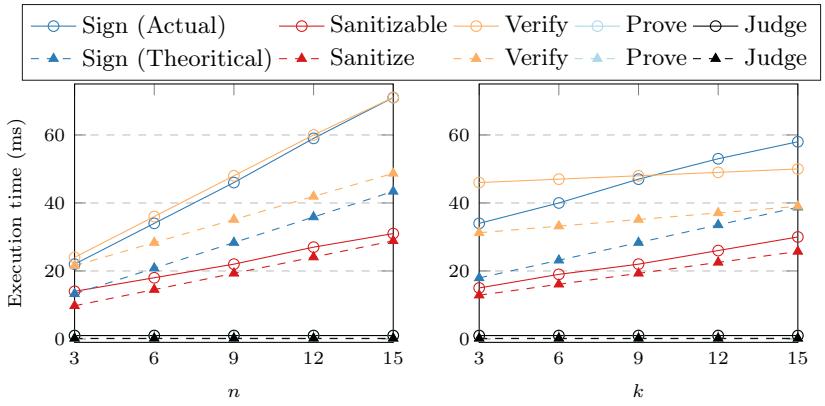

Fig. 10: Theoretical and real performance of IUT-k-SAN on the server. In the first figure, we fix $k = 9$ and test with variable n. In the second one, we fix $n = 9$ and test k.

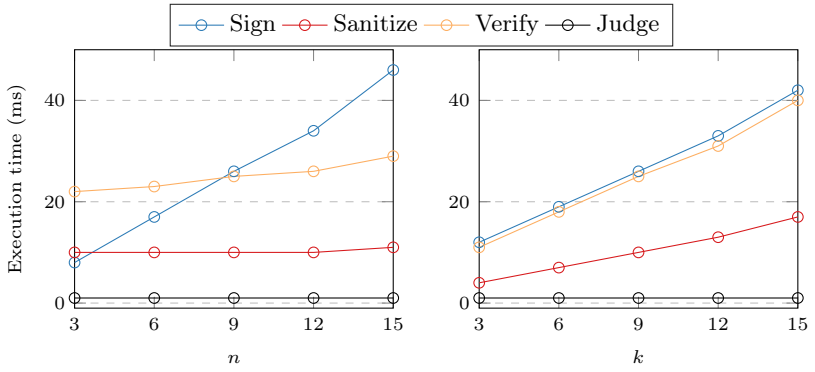

Fig. 11: Performance of FSV-k-SAN on the smartphone. In the first figure, we fix $k = 9$ and test with variable n. In the second one, we fix $n = 9$ and test k.

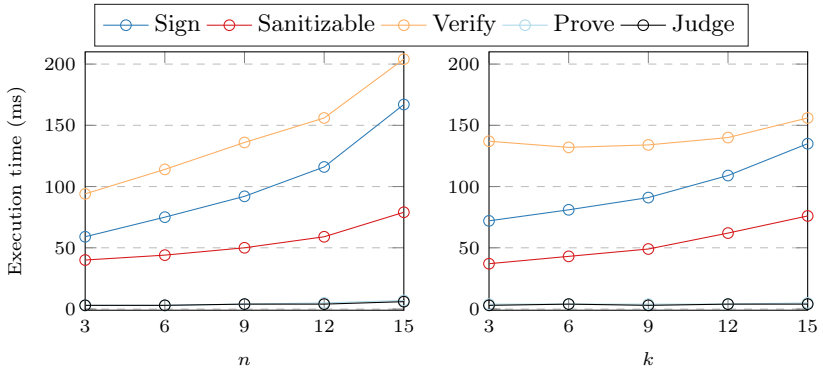

Fig. 12: Performance of IUT-k-SAN on the smartphone. In the first figure, we fix $k = 9$ and test with variable n. In the second one, we fix $n = 9$ and test k.

Table 3: Comparison of the efficiency of our k-SAN constructions and other multi-sanitizer schemes in terms of the signature size σ, the proof size π, and the cost of the main algorithms. n is the number of message blocks, k the number of sanitizers, t a polynomial in the size of the access structure of the attribute-based signature and the policy-based Chameleon hash used in [1] and [30] respectively, and Q is the number of previously generated signatures by the signer. We measure the size by the number of group elements and the execution time of algorithms by the number of modular exponentiations and pairing operations which are denoted by $\mathcal{E}$ and $\mathcal{P}$ respectively. Some algorithms do a number of operations that is equal to the number of modified, admissible, or inadmissible blocks. In these cases, we always consider the worst case which is n. The Sign algorithm of [30] generates RSA keys which is denoted by RSAGen.

Const.	$\lvert\sigma\rvert$	$\lvert\pi\rvert$	Sign	Sanitize	Verify	Prove	Judge
IUT-k-SAN	$O(kn)$	$O(1)$	$O(kn)\ \mathcal{E}$	$O(kn)\ \mathcal{E}$	$O(k)\ \mathcal{E}+$ $O(n)\ \mathcal{P}$	$O(1)\ \mathcal{E}$	$O(1)\ \mathcal{E}$
FSV-k-SAN	$O(kn)$	-	$O(kn)\ \mathcal{E}$	$O(kn)\ \mathcal{E}$	$O(kn)\ \mathcal{E}$	-	-
[1]	$O(t)$	$O(1)$	$O(t)\ \mathcal{E}$	$O(t)\ \mathcal{E}$	$O(1)\ \mathcal{E}+$ $O(t)\ \mathcal{P}$	$O(1)\ \mathcal{E}$	$O(1)\ \mathcal{P}$
[7]	$O(1)$	-	$O(1)\ \mathcal{E}$	$O(1)\ \mathcal{E}$	$O(1)\ \mathcal{E}$	-	$O(1)\ \mathcal{E}$
[15]	$O(n)$	-	$O(n)\ \mathcal{E}$	$O(n)\ \mathcal{E}$	$O(n)\ \mathcal{E}$	-	-
[25]	$O(n)$	$O(n)$	$O(n)\ \mathcal{E}+$ $O(n)\ \mathcal{P}$	$O(n)\ \mathcal{E}+$ $O(n)\ \mathcal{P}$	$O(n)\ \mathcal{E}+$ $O(n)\ \mathcal{P}$	$O(nQ)\ \mathcal{E}+$ $O(nQ)\ \mathcal{P}$	$O(n)\ \mathcal{E}+$ $O(n)\ \mathcal{P}$
[30]	$O(t)$	$O(1)$	RSAGen+ $O(t)\ \mathcal{E}$	$O(t)\ \mathcal{E}+$ $O(t)\ \mathcal{P}$	$O(1)\ \mathcal{E}+$ $O(1)\ \mathcal{P}$	$O(1)\ \mathcal{E}$	$O(1)\ \mathcal{E}$

either only provided a generic construction or mentioned concrete instantiations briefly without fully describing them. Thus, we provide an asymptotic complexity analysis for the signature and proof size and the execution time of the main algorithms. We measure size by the number of group elements and the execution time of algorithms by the number of modular exponentiations and pairing operations. Table 3 summarizes our results.

The signature of [1] consists of two signatures a randomizable signature for the inadmissible part of the message and an attribute-based one for the admissible part. Thus, their efficiency is linear in the size of the access policy of the attribute-based signature. The authors of [7] also use two normal signatures for the inadmissible and admissible parts of the message, thus they have constant size and execution time. The construction of [14] uses group signatures where we have a group for signers and a another group created per signature. It also uses non-interactive zero-knowledge proofs. The authors do not provide a concrete recommendation of which group signature or zero-knowledge proof to use which makes the performance evaluation require a significant effort as we have to choose a zero-knowledge proof and group signatures that are compatible and satisfy all the required security properties. The construction of [15] uses an identity-based Chameleon hash function to generate a hash of the admissible blocks and then sign the hash and the inadmissible blocks using a standard signature scheme. Their size and execution time is linear in the number of blocks. In [25], the construction follows the same steps as [15] but the identity-based chameleon hash is replaced with an accountable chameleon hash. Their signature size and execution time is also linear in the number of blocks but their Prove algorithm requires the signer to keep track of previously generated signatures and to do comparisons with then, which is inefficient. Finally, in [30], the signer computes a policy-based chameleon hash of the admissible part of the message, signs the hash, inadmissible part of the message, and other information, encrypts his public key, and generates a proof. The efficiency of the scheme is linear in the size of the access policy of the policy-based chameleon hash. Moreover, their Sign algorithm requires generating RSA keys which causes a significant performance overhead.

Our constructions' efficiency is linear in kn where k is the number of sanitizers and n is the number of message blocks. This is less efficient than most of the existing schemes, but this performance cost was necessary to support more security properties and to allow different admissibility policies for different sanitizers. Supporting different admissibility policies in the existing works will indeed lead to a significant loss in performance. For instance, for [1], we need to produce an attribute-based signatures for each set of blocks that are admissible to different access policies. The scheme of [7] would require a different signature per set of admissible blocks that are authorized to a certain sanitizer. The constructions of [15, 25, 30] will require multiple chameleon hashes under different keys or access policies for each set of blocks that are admissible to different sanitizers. Finally, the scheme of [14] would require creating multiple groups in the same

fashion. Moreover, some constructions might lead to worse performance even in the unique admissibility context. For example, the Sign algorithm of [30] requires generating RSA keys and the complexity of the Prove algorithm of [25] is linear in the number of previously generated signatures. Furthermore, the Verify algorithm of [1] does $32l + 80$ pairing operations where $l \times t$ is the size of claim-predicate monotone span program of the attribute based signature. If we compare this to IUT-k-SAN, we do $4n + 10$ pairing operations which could be more efficient depending on the number of blocks and [1]'s access policy.