Sanitizable Signatures with Different Admissibility Policies for Multiple Sanitizers

Osama Allabwani
osama.allabwani@limos.fr
Université Clermont Auvergne,
LIMOS, CNRS
Clermont-Ferrand, France
BeYs
Clermont-Ferrand, France

Olivier Blazy
olivier.blazy@polytechnique.edu
École Polytechnique
Palaiseau, France

Pascal Lafourcade
pascal.lafourcade@uca.fr
Université Clermont Auvergne,
LIMOS, CNRS
Clermont-Ferrand, France
ASTEROIDE, Trust4Sign
La Monnerie-le-Montel, France

Charles Olivier-Anclin
charles.olivier-anclin@uca.fr
Université Clermont Auvergne,
LIMOS, CNRS
Clermont-Ferrand, France

Olivier Raynaud
olivier.raynaud@isima.fr
Université Clermont Auvergne,
LIMOS, CNRS
Clermont-Ferrand, 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.

CCS Concepts

• **Security and privacy → Digital signatures**.

Keywords

Sanitizable Signatures, Multi Sanitizers, Public Key, Generic Construction, Privacy

ACM Reference Format:
Osama Allabwani, Olivier Blazy, Pascal Lafourcade, Charles Olivier-Anclin, and Olivier Raynaud. 2026. Sanitizable Signatures with Different Admissibility Policies for Multiple Sanitizers. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '26), June 1–5, 2026, Bangalore, India.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3779208.3785265

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASIA CCS '26, Bangalore, India
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-2356-8/26/06
https://doi.org/10.1145/3779208.3785265

1 Introduction

Sanitizable signatures introduced by Ateniese *et al.* [3] 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 [3], *accountability* which ensures that the signer or sanitizer cannot accuse the other party of signing a certain message [3], *privacy* which requires that the original message does not leak from the sanitized signature [3], *transparency* which guarantees indistinguishability between sanitized and fresh signatures [3], *invisibility* which aims to keep the class of admissible blocks hidden from external parties [3], and *unlinkability* which prevents linking sanitized signatures to their sources [9]. 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, 8, 15, 16, 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 [15]. 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 [3]. 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.* [12] by using two types of re-randomizable signatures [12, 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 [9, 12], 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 [12] 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, 8, 15, 16, 25, 30] and in particular Canard *et al.*'s work [15] 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 [15]. The construction does not satisfy full-sanitization verifiability because it breaks proof-restricted transparency and invisibility as explained before.

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

	Unforgeability	Immutability	Privacy	Accountability	Sanitizer Anonymity	Unlinkability	Full-Sanitization Verifiability	Transparency	Invisibility
FSV-k-SAN	✓	✓	✓	✓	✓	✗	✓	✗	✗
IUT-k-SAN	✓	✓	✓	✓	✓	✓	✗	✓	✓
[1]	✓	✓	✓	✓	✗	✓	✗	✓	✗
[8]	✓	✓	✓	✓	✗	✗	✗	✗	✗
[15]	✓	✓	✓	✓	✓	✓	✗	✓	✗
[16]	✓	✓	✓	✗	✗	✗	✗	✓	✗
[25]	✓	✓	✓	✓	✗	✗	✗	✓	✗
[30]	✓	✓	✓	✓	✓	✗	✗	✓	✗

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 [13], the authors 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 [3], who identified several security properties (unforgeability, immutability, privacy, transparency, and accountability) later formally defined in [7]. Invisibility was also introduced in [3] but received formal treatment much later in [14]. Unlinkability has been introduced and formalized in [9] and studied in [10, 21]. Only three schemes guarantee all these properties at once [6, 12, 13]. Two kinds of sanitizer limitation have also been considered in the literature encompassing limits in the number of sanitizations [13] or blocks sanitized in a single sanitization [6]. Sanitizable signatures with several sanitizers have been considered in multiple works [1, 8, 15, 16, 25, 30] but with only one admissibility policy common to all sanitizers. This setting required also defining new properties such as sanitizer anonymity [15]. 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, $[\![k]\!] = \{1, \dots, k\}$. 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 O and an algorithm $\mathcal{A}$, $\mathcal{A}^O$ means that the algorithm $\mathcal{A}$ has access to 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, $(a, b) \overset{\mathsf{p}}{\leftarrow} v$ 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 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 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 2.1 (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 pk_S, a list of sanitizer public keys **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 [12]. 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,
- $\mathsf{ADM}^{\mathbf{A}}_{\mathsf{SAN}}(m, m', i) = \bigwedge_{j=1}^{n} \left((m_j = m'_j) \vee a_{i,j} \right)$ — all modifications are allowed for sanitizer i,
- $\mathsf{ADM}^{\mathbf{A}}_{\mathsf{SET}}(m, m', \mathbf{Z}) = \bigwedge_{j=1}^{n} \left((m_j = m'_j) \vee (\exists i \in \mathbf{Z} \mid a_{i,j}) \right)$ — each change is permitted for some sanitizer in **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 **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 3.1 (Immutability [12]). A k-sanitizer sanitizable signature scheme k-SAN is said to be immutable if for all $k \geq 1$ and PPT adversaries $\mathcal{A}$, $\Pr\left[\mathsf{Exp}^{\mathsf{Immut},k}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ 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 $(\mathsf{pk}^*_S, m^*, \sigma^*, \pi^*)$ 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 3.2 (Sanitizer-Accountability [12]). A k-sanitizer sanitizable signature scheme k-SAN is said to be sanitizer-accountable if for all PPT adversaries $\mathcal{A}$, $\Pr\left[\mathsf{Exp}^{\mathsf{SanAcc}}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ where $\mathsf{Exp}^{\mathsf{SanAcc}}_{\text{k-SAN},\mathcal{A}}(\lambda)$ is defined in Figure 2.

Definition 3.3 (Signer-Accountability [12]). A k-sanitizer sanitizable signature scheme k-SAN is said to be signer-accountable if for all $k \geq 1$ and PPT adversaries $\mathcal{A}$, $\Pr\left[\mathsf{Exp}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ where $\mathsf{Exp}^{\mathsf{SigAcc},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$ is defined in Figure 2.

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 3.4 (Unforgeability [3]). A k-sanitizer sanitizable signature scheme k-SAN is said to be unforgeable if for all $k \geq 1$

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

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_S^\dagger, \mathbf{PKZ}, m, \mathbf{A})$
$\Sigma \coloneqq \Sigma \cup \{(\mathsf{pk}_S^\dagger, \mathbf{PKZ}, m, \mathbf{A}, \sigma)\}$
return σ

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

if $\mathsf{pk}_S \neq \mathsf{pk}_S^\dagger$ **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 $\bot$
  $\sigma' \leftarrow \mathsf{Sanitize}(\mathsf{sk}_Z^{\dagger,i}, \mathsf{pk}_S, \mathbf{PKZ}^\dagger, m, \mathsf{MOD}, \sigma)$
  return σ'
elseif $\left(\begin{array}{c}\exists(m', \sigma', \mathbf{A}') \in \Sigma \mid \\ m' = m \wedge \sigma' = \sigma \wedge \\ \mathsf{ADM}_{\mathsf{SAN}}^{\mathbf{A}'}(m, \mathsf{MOD}(m), i)\end{array}\right)$ **then**
  $\sigma' \leftarrow \mathsf{Sanitize}(\mathsf{sk}_Z^{\dagger,i}, \mathsf{pk}_S, \mathbf{PKZ}^\dagger, m, \mathsf{MOD}, \sigma)$
  $\Sigma \coloneqq \Sigma \cup \{(\mathsf{pk}_S, \mathbf{PKZ}^\dagger, \mathsf{MOD}(m), \mathbf{A}', \sigma')\}$
  return σ'
return $\bot$

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

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**
  $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_S^\dagger, \mathbf{PKZ}, m, \mathbf{A}_b)$
  if $\mathbf{PKZ} = \mathbf{PKZ}^\dagger$ **then**
    $\Sigma \coloneqq \Sigma \cup \{(\mathsf{pk}_S^\dagger, \mathbf{PKZ}, m, \mathbf{A}_0 \wedge \mathbf{A}_1, \sigma)\}$
  return σ
return $\bot$

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

$b_\beta \leftarrow \mathsf{Verify}(\mathsf{pk}_S^\dagger, \mathbf{PKZ}^\dagger, m_\beta, \sigma_\beta), \forall \beta \in \{0,1\}$
if $\neg b_0 \vee \neg b_1$ **then return** $\bot$
foreach $\beta \in \{0,1\}$ **do**
  $\mathbf{A}_\beta \leftarrow \mathsf{ExtA}(\mathsf{sk}_Z^{\dagger,i_\beta}, \sigma_\beta)$
  $\sigma'_\beta \leftarrow \mathsf{Sanitize}(\mathsf{sk}_Z^{\dagger,i_\beta}, \mathsf{pk}_S^\dagger, \mathbf{PKZ}^\dagger, m_\beta, \mathsf{MOD}_\beta, \sigma_\beta)$
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**
  $\Sigma \coloneqq \Sigma \cup \{(\mathsf{pk}_S^\dagger, \mathbf{PKZ}^\dagger, \mathsf{MOD}_b(m_b), \mathbf{A}_b, \sigma'_b)\}$
  return σ'_b
return $\bot$

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

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

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

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

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

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

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

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

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

and PPT adversaries $\mathcal{A}$, $\Pr\left[\mathsf{Exp}^{\mathsf{Unforg},k}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ where $\mathsf{Exp}^{\mathsf{Unforg},k}_{\text{k-SAN},\mathcal{A}}(\lambda)$ is defined in Figure 2.

As established in [7, 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 $\bot$ on signatures generated via $\mathcal{O}^b_{\mathsf{Sign/Sanit}}$.

Definition 3.5 (Proof-Restricted Transparency [12]). A k-sanitizer sanitizable signature scheme k-SAN is said to be proof-restricted transparent if for all $k \geq 1$ and PPT adversaries $\mathcal{A}$,
$\Pr\left[\mathsf{Exp}^{\mathsf{Trans},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ 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 $\{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{Sanit}}, \mathcal{O}_{\mathsf{Prove}}\}$.

$\mathsf{KGenAll}(k)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 2: Security Experiments for k-SAN.

Definition 3.6 (Privacy [3]). A k-sanitizer sanitizable signature scheme k-SAN is said to be private if for all $k \geq 1$ and PPT adversaries $\mathcal{A}$, $\Pr\left[\mathsf{Exp}^{\mathsf{Priv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ 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 [7, 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 $\{O_{\mathsf{Sanit}'}, O^b_\mathsf{LoRADM}, O_\mathsf{Prove}\}$.

Definition 3.7 (Invisibility [12]). A k-sanitizer sanitizable signature scheme k-SAN is said to be invisible if for all $k \geq 1$ and PPT adversaries $\mathcal{A}$, $\Pr\left[\mathsf{Exp}^{\mathsf{Inv},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ 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 $\{O_\mathsf{Sign}, O_\mathsf{Sanit}, O^b_\mathsf{LoRSanit}, O_\mathsf{Prove}\}$ in order to win.

Definition 3.8 (Unlinkability [12]). A k-sanitizer sanitizable signature scheme k-SAN is said to be unlinkable if for all $k \geq 1$ and PPT adversaries $\mathcal{A}$, $\Pr\left[\mathsf{Exp}^{\mathsf{Unlink},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ 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 $\mathbf{PKZ}$, 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 3.9 (Sanitizer Anonymity [15]). A k-sanitizer sanitizable signature scheme k-SAN is said to be sanitizer anonymous if for all $k \geq 2$ and PPT adversaries $\mathcal{A}$,
$\Pr\left[\mathsf{Exp}^{\mathsf{SanitAnon},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1 \vee \mathsf{Exp}^{\mathsf{AdmSec},k,b}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$
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 $(\mathsf{pk}^*_S, m^*, \sigma^*)$ 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 3.10 (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 $\mathcal{A}$, $\Pr\left[\mathsf{Exp}^{\mathsf{FullSanit},k}_{\text{k-SAN},\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$ 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 3.11. *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 $(\mathsf{pp}, \mathsf{pk}^\dagger_S, \mathbf{PKZ}^\dagger)$ 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 $\mathcal{O}^b_{\mathsf{Sign/Sanit}}(\mathsf{pk}^{\dagger,i}_Z, m, \mathsf{MOD}, \mathbf{A})$ oracle which will return a signature σ. Finally, $\mathcal{A}$ calls $\mathsf{Verify}(\mathsf{pk}^\dagger_S, \mathbf{PKZ}^\dagger, m, \sigma)$ 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 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 3.12. *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 $(\mathsf{pp}, \mathsf{pk}^\dagger_S, \mathbf{PKZ}^\dagger)$ 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 $\mathsf{Verify}(\mathsf{pk}^\dagger_S, \mathbf{PKZ}^\dagger, m, \sigma)$ 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 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 which are differed to the full version [2].

Chameleon Hash functions which were introduced in [23] are commonly used in sanitizable signatures. They are described as $\mathsf{CHash} := (\mathsf{ParGen}_{\mathsf{CHash}}, \mathsf{KGen}_{\mathsf{CHash}}, \mathsf{Hash}_{\mathsf{CHash}}, \mathsf{Check}_{\mathsf{CHash}}, \mathsf{Adapt}_{\mathsf{CHash}})$. 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 $\mathsf{SIG} := (\mathsf{KGen}_{\mathsf{SIG}}, \mathsf{Sign}_{\mathsf{SIG}}, \mathsf{Verify}_{\mathsf{SIG}})$ and $\mathsf{PKE} := (\mathsf{KGen}_{\mathsf{PKE}}, \mathsf{Encrypt}_{\mathsf{PKE}}, \mathsf{Decrypt}_{\mathsf{PKE}}, \mathsf{Multiply}_{\mathsf{PKE}})$ 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 $\mathsf{VRS} := (\mathsf{Setup}_{\mathsf{VRS}}, \mathsf{KGen}_{\mathsf{VRS}}, \mathsf{Sign}_{\mathsf{VRS}}, \mathsf{Verify}_{\mathsf{VRS}}, \mathsf{Prove}_{\mathsf{VRS}}, \mathsf{Judge}_{\mathsf{VRS}})$. 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 [11] 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 EQS $:=$ (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 [12] 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 [12]. Each key is used to sign a message block m_j after applying a hash function $\mathcal{H}(j\|m_j) \rightarrow \mathbb{G}_2$. 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 **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 Sanitize algorithm, the sanitizer decrypts his ciphertexts in **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 **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 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.* [12] as a baseline, and then we did the necessary changes to satisfy the requirements of k-SAN. Notably, in [12], 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 5.1. 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 which is differed to the full version [2]. 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 5.2. *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:

$\mathsf{Setup}(\lambda, n)$

$\mathcal{BG} \leftarrow \mathsf{BGGen}(\lambda), \mathsf{pp}_{\mathsf{VRS}} \leftarrow \mathsf{Setup}_{\mathsf{VRS}}(\lambda)$
$\mathsf{pp} := (\mathcal{BG}, \lambda, n, \mathsf{pp}_{\mathsf{VRS}})$
return pp

$\mathsf{Sign}(\mathsf{sk_S}, \mathbf{PKZ}, m, \mathbf{A})$

$\mathbf{SKZ} := \perp, (\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}})_{i \in [\![k]\!]} \overset{\mathsf{p}}{\leftarrow} \mathbf{PKZ}$
$(\mathsf{sk_{EQS}}, \mathsf{sk_{SP}}) \overset{\mathsf{p}}{\leftarrow} \mathsf{sk_S}, (\mathsf{pk_{EQS}}, \mathsf{pk_{SP}}) \overset{\mathsf{p}}{\leftarrow} \mathsf{pk_S}$
$m := m \| \mathbf{PKZ}, \mathbf{A} \leftarrow \mathsf{AppendC}(\mathbf{A}, (0)^{[\![k]\!]})$
$x_j, y_j \leftarrow\$ \mathbb{Z}_q^*, X_j := G_1^{x_j}, Y_j := X_j^{y_j}, \forall j \in [\![n]\!]$
$\mu := \mathsf{Sign_{EQS}}(\mathsf{sk_{EQS}}, (X_j)_{j \in [\![n]\!]})$
$\eta := \mathsf{Sign_{EQS}}(\mathsf{sk_{EQS}}, (Y_j)_{j \in [\![n]\!]})$
$\sigma_j := \mathcal{H}(j \| m_j)^{y_j}, \forall j \in [\![n]\!]$
foreach $j \in [\![n]\!]$ **do**
  $\mathbf{T} := \perp$
  foreach $i \in [\![k]\!]$ **do**
    $\tau \leftarrow \mathsf{Encrypt_{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, y_j \cdot a_{i,j})$
    $\mathbf{T} \leftarrow \mathsf{Append}(\mathbf{T}, \tau)$
  $\mathbf{SKZ} \leftarrow \mathsf{AppendC}(\mathbf{SKZ}, \mathbf{T})$
$\sigma_{\mathsf{SS}} := (\mu, \eta, (\sigma_j, X_j, Y_j)_{j \in [\![n]\!]}, \mathbf{SKZ})$
$t := \mathsf{pk_S} \| m \| \sigma_{\mathsf{SS}}, L := \{\mathsf{pk_{SP}}\} \cup \{\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]}$
$\sigma_{\mathsf{VRS}} \leftarrow \mathsf{Sign_{VRS}}(\mathsf{sk_{SP}}, L, t)$
$\sigma := (\sigma_{\mathsf{SS}}, \sigma_{\mathsf{VRS}})$
return σ

$\mathsf{Prove}(\mathsf{sk_S}, \mathbf{PKZ}, m, \sigma, j)$

$(\mathsf{sk_{EQS}}, \mathsf{sk_{SP}}) \overset{\mathsf{p}}{\leftarrow} \mathsf{sk_S}, (\mathsf{pk_{EQS}}, \mathsf{pk_{SP}}) \overset{\mathsf{p}}{\leftarrow} \mathsf{pk_S}$
$(\sigma_{\mathsf{SS}}, \sigma_{\mathsf{VRS}}) \overset{\mathsf{p}}{\leftarrow} \sigma$
$(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}})_{i \in [\![k]\!]} \overset{\mathsf{p}}{\leftarrow} \mathbf{PKZ}, m := m \| \mathbf{PKZ}$
$t := \mathsf{pk_S} \| m \| \sigma_{\mathsf{SS}}, L := \{\mathsf{pk_{SP}}\} \cup \{\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]}$
$\pi \leftarrow \mathsf{Prove_{VRS}}(L, t, \sigma_{\mathsf{VRS}}, \mathsf{pk_{SP}}, \mathsf{sk_{SP}})$
return π

$\mathsf{Sanitize}(\mathsf{sk_Z}, \mathsf{pk_S}, \mathbf{PKZ}, m, \mathsf{MOD}, \sigma)$

$(\mathsf{sk_{ZE}}, \mathsf{sk_{ZP}}) \overset{\mathsf{p}}{\leftarrow} \mathsf{sk_Z}, (\mathsf{pk_{EQS}}, \mathsf{pk_{SP}}) \overset{\mathsf{p}}{\leftarrow} \mathsf{pk_S}$
$(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}})_{i \in [\![k]\!]} \overset{\mathsf{p}}{\leftarrow} \mathbf{PKZ}, (\sigma_{\mathsf{SS}}, \sigma_{\mathsf{VRS}}) \overset{\mathsf{p}}{\leftarrow} \sigma$
$(\mu, \eta, (\sigma_j, X_j, Y_j)_{j \in [\![n]\!]}, \mathbf{SKZ}) \overset{\mathsf{p}}{\leftarrow} \sigma_{\mathsf{SS}}$
$m := m \| \mathbf{PKZ}, m' = \mathsf{MOD}(m)$
$r, s \leftarrow\$ \mathbb{Z}_q^*, \mathbf{SKZ}' := \perp$
$i' := \{i \in [\![k]\!] \mid \mathsf{pk_Z} = (\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}})\}$
$(X'_j)_{j \in [\![n]\!]} := (X_j^r)_{j \in [\![n]\!]}$
$(Y'_j)_{j \in [\![n]\!]} := (Y_j^{r \cdot s})_{j \in [\![n]\!]}$
$\mu' \leftarrow \mathsf{ChgRep_{EQS}}(\mathsf{pk_{EQS}}, (X_j)_{j \in [\![n]\!]}, \mu, r)$
$\eta' \leftarrow \mathsf{ChgRep_{EQS}}(\mathsf{pk_{EQS}}, (Y_j)_{j \in [\![n]\!]}, \eta, r \cdot s)$
foreach $j \in [\![n]\!]$ **do**
  if $j \in \mathsf{MOD}$ **do**
    $\zeta \leftarrow \mathsf{Decrypt_{PKE}}(\mathsf{sk_{ZE}}, \mathbf{SKZ}_{i',j})$
    return $\perp$ **if** $\zeta = 0$
    $\sigma'_j := \mathcal{H}(j \| m'_j)^{\zeta \cdot s}$
  else
    $\sigma'_j := \sigma_j^s$
  $\mathbf{T} := \perp$
  foreach $i \in [\![k]\!]$ **do**
    $\tau \leftarrow \mathsf{Multiply_{PKE}}(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathbf{SKZ}_{i,j}, s)$
    $\mathbf{T} \leftarrow \mathsf{Append}(\mathbf{T}, \tau)$
  $\mathbf{SKZ}' \leftarrow \mathsf{AppendC}(\mathbf{SKZ}', \mathbf{T})$
$\sigma'_{\mathsf{SS}} := (\mu', \eta', (\sigma'_j, X'_j, Y'_j)_{j \in [\![n]\!]}, \mathbf{SKZ}')$
$t := \mathsf{pk_S} \| m' \| \sigma'_{\mathsf{SS}}, L := \{\mathsf{pk_{SP}}\} \cup \{\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]}$
$\sigma'_{\mathsf{VRS}} \leftarrow \mathsf{Sign_{VRS}}(\mathsf{sk_{ZP}}, L, t)$
$\sigma' := (\sigma'_{\mathsf{SS}}, \sigma'_{\mathsf{VRS}})$
return σ'

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

$(\mathsf{sk_{EQS}}, \mathsf{pk_{EQS}}) \leftarrow \mathsf{KGen_{EQS}}(\mathcal{BG}, n)$
$(\mathsf{sk_{SP}}, \mathsf{pk_{SP}}) \leftarrow \mathsf{KGen_{VRS}}(\mathsf{pp_{VRS}})$
$\mathsf{sk_S} := (\mathsf{sk_{EQS}}, \mathsf{sk_{SP}}), \mathsf{pk_S} := (\mathsf{pk_{EQS}}, \mathsf{pk_{SP}})$
return $(\mathsf{sk_S}, \mathsf{pk_S})$

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

$(\mathsf{sk_{ZE}}, \mathsf{pk_{ZE}}) \leftarrow \mathsf{KGen_{PKE}}(\lambda)$
$(\mathsf{sk_{ZP}}, \mathsf{pk_{ZP}}) \leftarrow \mathsf{KGen_{VRS}}(\mathsf{pp_{VRS}})$
$\mathsf{sk_Z} := (\mathsf{sk_{ZE}}, \mathsf{sk_{ZP}}), \mathsf{pk_Z} := (\mathsf{pk_{ZE}}, \mathsf{pk_{ZP}})$
return $(\mathsf{sk_Z}, \mathsf{pk_Z})$

$\mathsf{Verify}(\mathsf{pk_S}, \mathbf{PKZ}, m, \sigma)$

$(\mathsf{pk_{EQS}}, \mathsf{pk_{SP}}) \overset{\mathsf{p}}{\leftarrow} \mathsf{pk_S}, (\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}})_{i \in [\![k]\!]} \overset{\mathsf{p}}{\leftarrow} \mathbf{PKZ}$
$(\sigma_{\mathsf{SS}}, \sigma_{\mathsf{VRS}}) \overset{\mathsf{p}}{\leftarrow} \sigma, m := m \| \mathbf{PKZ}$
$(\mu, \eta, (\sigma_j, X_j, Y_j)_{j \in [\![n]\!]}, \mathbf{SKZ}) \overset{\mathsf{p}}{\leftarrow} \sigma_{\mathsf{SS}}$
$t := \mathsf{pk_S} \| m \| \sigma_{\mathsf{SS}}, L := \{\mathsf{pk_{SP}}\} \cup \{\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]}$
$b_{-3} \leftarrow \mathsf{Verify_{VRS}}(L, t, \sigma_{\mathsf{VRS}})$
$b_{-2} := (\forall j \in [\![n]\!], Y_j \neq G_1)$
$b_{-1} \leftarrow \mathsf{Verify_{EQS}}(\mathsf{pk_{EQS}}, (X_j)_{j \in [\![n]\!]}, \mu)$
$b_0 \leftarrow \mathsf{Verify_{EQS}}(\mathsf{pk_{EQS}}, (Y_j)_{j \in [\![n]\!]}, \eta)$
$b_j := (\forall j \in [\![n]\!], (e(X_j, \sigma_j) = e(Y_j, \mathcal{H}(j \| m_j))))$
return $\bigwedge_{j=-3}^{n} b_j$

$\mathsf{Judge}(\mathsf{pk_S}, \mathbf{PKZ}, m, \sigma, \pi, j)$

$(\mathsf{pk_{EQS}}, \mathsf{pk_{SP}}) \overset{\mathsf{p}}{\leftarrow} \mathsf{pk_S}, (\sigma_{\mathsf{SS}}, \sigma_{\mathsf{VRS}}) \overset{\mathsf{p}}{\leftarrow} \sigma$
$(\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZE}}, \mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}})_{i \in [\![k]\!]} \overset{\mathsf{p}}{\leftarrow} \mathbf{PKZ}, m := m \| \mathbf{PKZ}$
$t := \mathsf{pk_S} \| m \| \sigma_{\mathsf{SS}}, L := \{\mathsf{pk_{SP}}\} \cup \{\mathsf{pk}^{\mathsf{i}}_{\mathsf{ZP}}\}_{i \in [\![k]\!]}$
$b \leftarrow \mathsf{Judge_{VRS}}(L, t, \sigma_{\mathsf{VRS}}, \mathsf{pk_{SP}}, \pi)$
return Z **if** $b = 0$ and S **if** $b = 1$

Figure 3: Algorithms of the IUT-k-SAN **construction.**

(1) distinguish whether the EQS signature was produced via $\mathsf{ChgRep_{EQS}}$ or directly with $\mathsf{Sign_{EQS}}$;
(2) tell whether the ciphertexts in **SKZ** were generated using $\mathsf{Multiply_{PKE}}$ or $\mathsf{Encrypt_{PKE}}$;
(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 O^b_{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 O^b_{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 $\mathsf{Decrypt_{PKE}}$. In the second hybrid, we stop using $\mathsf{Multiply_{PKE}}$ in the 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 $\mathsf{ChgRep_{EQS}}$ in the Sanitize algorithm and we use $\mathsf{Sign_{EQS}}$ 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.,* [14]).

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 **CH**. A matrix of ciphertexts **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 **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 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 6.1. 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 which is differed to the full version [2]. 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

$\mathsf{Setup}(\lambda, n)$

$pp_{CH} \leftarrow \mathsf{ParGen}_{\mathsf{CHash}}(\lambda)$
$pp_{VRS} \leftarrow \mathsf{Setup}_{\mathsf{VRS}}(\lambda)$
$pp := (pp_{CH}, pp_{VRS})$
return pp

$\mathsf{Sign}(sk_S, \mathbf{PKZ}, m, \mathbf{A})$

$\mathbf{SKCH} := \bot, \mathbf{CH} := \bot, \rho := \bot, n := |m|$
$(pk^i_{ZE}, pk^i_{ZP})_{i \in [\![k]\!]} \overset{p}{\leftarrow} \mathbf{PKZ}$
foreach $j \in [\![n]\!]$ **do**
  $(sk_{CH}, pk_{CH}) \leftarrow \mathsf{KGen}_{\mathsf{CHash}}(pp)$
  $(h, r) \leftarrow \mathsf{Hash}_{\mathsf{CHash}}(pk_{CH}, (j \| m_j))$
  $\mathbf{CH} \leftarrow \mathsf{Append}(\mathbf{CH}, (h, r, pk_{CH}))$
  $\mathbf{T} := \bot$
  foreach $i \in [\![k]\!]$ **do**
    $\tau \leftarrow \mathsf{Encrypt}_{\mathsf{PKE}}(pk^i_{ZE}, sk_{CH} \cdot a_{i,j})$
    $\mathbf{T} \leftarrow \mathsf{Append}(\mathbf{T}, \tau)$
  $\mathbf{SKCH} \leftarrow \mathsf{AppendC}(\mathbf{SKCH}, \mathbf{T})$
$\mathbf{PA} := (\mathsf{ADM}^{\mathbf{A}}(j))_{j \in [\![n]\!]}$
$ms := \begin{pmatrix} (\mathbf{CH}_j.h, \mathbf{CH}_j.pk_{CH})_{j \in [\![n]\!]}, \mathbf{SKCH}, \\ \mathbf{PA}, pk_S, \mathbf{PKZ}, n \end{pmatrix}$
$s \leftarrow \mathsf{Sign}_{\mathsf{SIG}}(sk_S, ms)$
foreach $j \in [\![n]\!]$ **do**
  $\tau \leftarrow \mathsf{Sign}_{\mathsf{SIG}}(sk_S, (j \| m_j \| s))$
  $\rho \leftarrow \mathsf{Append}(\rho, \tau)$
$\sigma := (s, \mathbf{CH}', \mathbf{SKCH}, \mathbf{PA}, n, \rho')$
return σ

$\mathsf{KGen}_Z(pp)$

$(sk_{ZE}, pk_{ZE}) \leftarrow \mathsf{KGen}_{\mathsf{PKE}}(\lambda)$
$(sk_{ZP}, pk_{ZP}) \leftarrow \mathsf{KGen}_{\mathsf{VRS}}(pp_{VRS})$
$sk_Z := (sk_{ZE}, sk_{ZP}), pk_Z := (pk_{ZE}, pk_{ZP})$
return (sk_Z, pk_Z)

$\mathsf{Sanitize}(sk_Z, pk_S, \mathbf{PKZ}, m, \mathsf{MOD}, \sigma)$

$\mathbf{CH}' := \bot, \rho' := \bot$
$(pk^i_{ZE}, pk^i_{ZP})_{i \in [\![k]\!]} \overset{p}{\leftarrow} \mathbf{PKZ}$
$(s, \mathbf{CH}, \mathbf{SKCH}, \mathbf{PA}, n, \rho) \overset{p}{\leftarrow} \sigma$
$(sk_{ZE}, sk_{ZP}) \overset{p}{\leftarrow} sk_Z, (h_j, r_j, pk^j_{CH})_{j \in [\![n]\!]} \overset{p}{\leftarrow} \mathbf{CH}$
$m' = \mathsf{MOD}(m), L := \{pk^i_{ZP}\}_{i \in [\![k]\!]}$
$i' := i \in [\![k]\!] \mid pk_Z = (pk^i_{ZE}, pk^i_{ZP})$
foreach $j \in [\![n]\!]$ **do**
  if $j \in \mathsf{MOD}$ **then**
    $\tau \leftarrow \mathsf{Decrypt}_{\mathsf{PKE}}(sk_{ZE}, \mathbf{SKCH}_{i',j})$
    $r' \leftarrow \mathsf{Adapt}_{\mathsf{CHash}}(\tau, (j \| m_j), (j \| m'_j), r_j, h_j)$
    if $r' = \bot$ **then return** $\bot$
    $\tau \leftarrow \mathsf{Sign}_{\mathsf{VRS}}(sk_{ZP}, L, (j \| m'_j, s))$
    $\rho' \leftarrow \mathsf{Append}(\rho', \tau)$
    $\mathbf{CH}' \leftarrow \mathsf{Append}(\mathbf{CH}', (h_j, r', pk^j_{CH}))$
  else
    $\rho' \leftarrow \mathsf{Append}(\rho', \rho_j)$
    $\mathbf{CH}' \leftarrow \mathsf{Append}(\mathbf{CH}', (h_j, r_j, pk^j_{CH}))$
$\sigma' := (s, \mathbf{CH}, \mathbf{SKCH}, \mathbf{PA}, n, \rho)$
return σ'

$\mathsf{KGen}_S(pp)$

$(sk_S, pk_S) \leftarrow \mathsf{KGen}_{\mathsf{SIG}}(\lambda)$
return (sk_S, pk_S)

$\mathsf{Verify}(pk_S, \mathbf{PKZ}, m, \sigma)$

$(s, \mathbf{CH}, \mathbf{SKCH}, \mathbf{PA}, n, \rho) \overset{p}{\leftarrow} \sigma$
$(h_j, r_j, pk^j_{CH})_{j \in [\![n]\!]} \overset{p}{\leftarrow} \mathbf{CH}$
$ms := \begin{pmatrix} (h_j, pk^j_{CH})_{j \in [\![n]\!]}, \mathbf{SKCH}, \\ \mathbf{PA}, pk_S, \mathbf{PKZ}, n \end{pmatrix}$
$b_1 \leftarrow \mathsf{Verify}_{\mathsf{SIG}}(pk_S, ms, s)$
$b_2 := \bigwedge_{j=1}^{n} \left\{ \mathsf{Check}_{\mathsf{CHash}}(pk^j_{CH}, (j \| m_j), r_j, h_j) \right\}$
$(pk^i_{ZE}, pk^i_{ZP})_{i \in [\![k]\!]} \overset{p}{\leftarrow} \mathbf{PKZ}, L := \{pk^i_{ZP}\}_{i \in [\![k]\!]}$
$b_3 := \bigwedge_{j=1}^{n} \left\{ \begin{array}{c} \left(\begin{array}{c} \neg \mathbf{PA}_j \wedge \\ \mathsf{Verify}_{\mathsf{SIG}}(pk_S, (j \| m_j \| s), \rho_j) = 1 \end{array} \right) \\ \vee \left(\begin{array}{c} \mathbf{PA}_j \wedge \\ \mathsf{Verify}_{\mathsf{VRS}}(L, (j \| m_j \| s), \rho_j) = 1 \end{array} \right) \end{array} \right\}$
return $b_1 \wedge b_2 \wedge b_3$

$\mathsf{Judge}(pk_S, \mathbf{PKZ}, m, \sigma, \pi, j)$

$(s, \mathbf{CH}, \mathbf{SKCH}, \mathbf{PA}, n, \rho) \overset{p}{\leftarrow} \sigma$
if $j = \bot$ **then**
  if $\exists j \in [\![n]\!], \mathbf{PA}_j = 1$ **then return** Z
  else return S
else
  if $\mathbf{PA}_j = 1$ **then return** Z
  else return S

Figure 4: Algorithms of the FSV-k-SAN **construction.**

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 [11], Mercurial signatures [18] 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.

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.

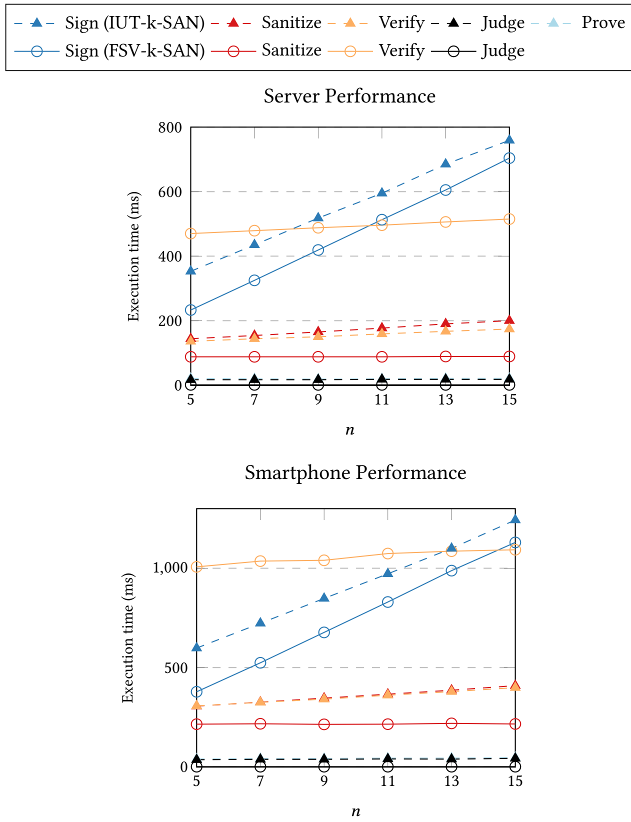

Figure 5: Performance of the implementation of both constructions. We fix $k = 5$ and test with variable n. The first figure shows the performance on a server, while the second figure shows the performance on a smartphone.

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 ~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.

We also compare the efficiency of our constructions with the existing multi-sanitizer schemes [1, 8, 15, 16, 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 [17], Kyber [5], and Dilithium [20]. Since the Chameleon hash of [17] 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 European Astral Project ERASMUS-EDU-2024:101184483 and the French National Research Agency through the PRIvacy-preserving tools for VAlidation and Security of Queries (PRIVASIQ) project (ANR-23-CE39-0008).

References

- [1] Ismail Afia and Riham AlTawy. 2023. Unlinkable Policy-Based Sanitizable Signatures. In *CT-RSA 2023*, Vol. 13871.
- [2] Osama Allabwani, Olivier Blazy, Pascal Lafourcade, Charles Olivier-Anclin, and Olivier Raynaud. 2025. Sanitizable Signatures with Different Admissibility Policies for Multiple Sanitizers. https://hal.science/hal-05411833
- [3] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. 2005. Sanitizable Signatures. In *ESORICS 2005*, Vol. 3679.
- [4] Nuttapong Attrapadung, Javier Herranz, Fabien Laguillaumie, Benoît Libert, Elie De Panafieu, and Carla Ràfols. 2012. Attribute-based encryption schemes with constant-size ciphertexts. *Theoretical computer science* 422 (2012).
- [5] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *EuroS&P 2018*.
- [6] Angèle Bossuat and Xavier Bultel. 2021. Unlinkable and Invisible γ-Sanitizable Signatures. In *ACNS 2021*, Vol. 12726.
- [7] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. 2009. Security of Sanitizable Signatures Revisited. In *PKC 2009*, Vol. 5443.
- [8] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. 2009. Sanitizable signatures: How to partially delegate control for authenticated data. (2009).

[9] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. 2010. Unlinkability of Sanitizable Signatures. In *PKC 2010*, Vol. 6056.
[10] Christina Brzuska, Henrich C Pöhls, and Kai Samelin. 2014. Efficient and perfectly unlinkable sanitizable signatures without group signatures. In *EuroPKI 2014*.
[11] Xavier Bultel and Pascal Lafourcade. 2017. Unlinkable and Strongly Accountable Sanitizable Signatures from Verifiable Ring Signatures. In *CANS 17*, Vol. 11261.
[12] Xavier Bultel, Pascal Lafourcade, Russell W. F. Lai, Giulio Malavolta, Dominique Schröder, and Sri Aravinda Krishnan Thyagarajan. 2019. Efficient Invisible and Unlinkable Sanitizable Signatures. In *PKC 2019, Part I*, Vol. 11442.
[13] Xavier Bultel and Charles Olivier-Anclin. 2024. Taming Delegations in Anonymous Signatures: k-Times Anonymity for Proxy and Sanitizable Signature. In *CANS 2024, Part I*, Vol. 14905.
[14] Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. 2017. Chameleon-Hashes with Ephemeral Trapdoors - And Applications to Invisible Sanitizable Signatures. In *PKC 2017, Part II*, Vol. 10175.
[15] Sébastien Canard, Amandine Jambert, and Roch Lescuyer. 2012. Sanitizable Signatures with Several Signers and Sanitizers. In *AFRICACRYPT 12*, Vol. 7374.
[16] Sébastien Canard, Fabien Laguillaumie, and Michel Milhau. 2008. Trapdoor sanitizable signatures and their application to content protection. In *ACNS 2008*.
[17] Sebastian Clermont, Samed Düzlü, Christian Janson, Laurens Porzenheim, and Patrick Struck. 2025. Lattice-Based Sanitizable Signature Schemes: Chameleon Hash Functions and More. In *PQCrypto 2025*.
[18] Elizabeth C. Crites and Anna Lysyanskaya. 2019. Delegatable Anonymous Credentials from Mercurial Signatures. In *CT-RSA 2019*, Vol. 11405.
[19] David Derler, Kai Samelin, Daniel Slamanig, and Christoph Striecks. 2019. Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based. In *NDSS 2019*.
[20] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018).
[21] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. 2016. Efficient Unlinkable Sanitizable Signatures from Signatures with Re-randomizable Keys. In *PKC 2016, Part I*, Vol. 9614.
[22] Christian Hanser and Daniel Slamanig. 2014. Structure-Preserving Signatures on Equivalence Classes and Their Application to Anonymous Credentials. In *ASIACRYPT 2014, Part I*, Vol. 8873.
[23] Hugo Krawczyk and Tal Rabin. 1997. Chameleon hashing and signatures. *Internet* (1997).
[24] Stephan Krenn, Kai Samelin, and Dieter Sommer. 2015. Stronger security for sanitizable signatures. In *DPM 2015*.
[25] Junzuo Lai, Xuhua Ding, and Yongdong Wu. 2013. Accountable trapdoor sanitizable signatures. In *ISPEC 2013*.
[26] Jiqiang Lv and X Wang. 2003. Verifiable ring signature. In *DMS 2003*.
[27] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT'99*, Vol. 1592.
[28] Manoj Prabhakaran and Mike Rosulek. 2008. Homomorphic encryption with CCA security. In *ICALP 2008*.
[29] Ronald L. Rivest, Adi Shamir, and Yael Tauman. 2001. How to Leak a Secret. In *ASIACRYPT 2001*, Vol. 2248.
[30] Kai Samelin and Daniel Slamanig. 2020. Policy-Based Sanitizable Signatures. In *CT-RSA 2020*, Vol. 12006.
[31] Claus-Peter Schnorr. 1991. Efficient Signature Generation by Smart Cards. *Journal of Cryptology* 4, 3 (1991).
[32] Michael Scott. 2019. Pairing Implementation Revisited. Cryptology ePrint Archive, Report 2019/077.

A Building Blocks Algorithms

This section is differed to the full version [2] due to space limitation.

B Security Experiments for The Building Blocks

This section is differed to the full version [2] due to space limitation.

C Proofs for The IUT-k-SAN Construction

This section is differed to the full version [2] due to space limitation.

D High-Level Flow Charts

Figures 7 and 6 show how the Sign and Sanitize algorithms work on a high-level in both constructions.

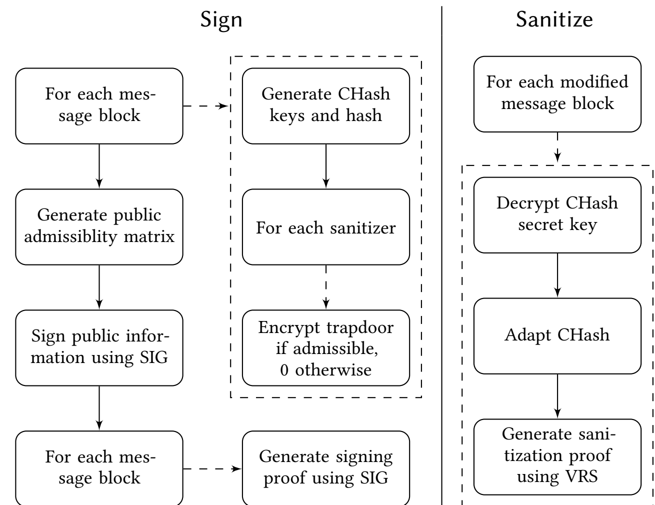

Figure 6: High-level flow chart showing how the Sign and Sanitize algorithms work in FSV-k-SAN. A dashed line indicates that the action is done inside a loop and a dashed box groups multiple actions that are done inside the same loop.

E Proofs for The FSV-k-SAN Construction

This section is differed to the full version [2] due to space limitation.

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 [19, 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 [4] which come with different trade-offs such as restrictions on the policies that can be defined and/or linear or quadratic secret key sizes.

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	$3n\,\mathbb{Z}_q + kn\,\mathbb{Z}_{N^2} + (n+1)\,\mathbb{G}_1$	$\lvert\mathsf{MOD}\rvert(4k-1)\,\mathbb{Z}_q + \lvert\mathsf{MOD}\rvert\,\mathbb{Z}_{N^2}$	$(4kn_1+2n)\,\mathbb{Z}_q + (2n_0+2)\,\mathbb{G}_1$	-	-

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

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 8 and 9 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 increase 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

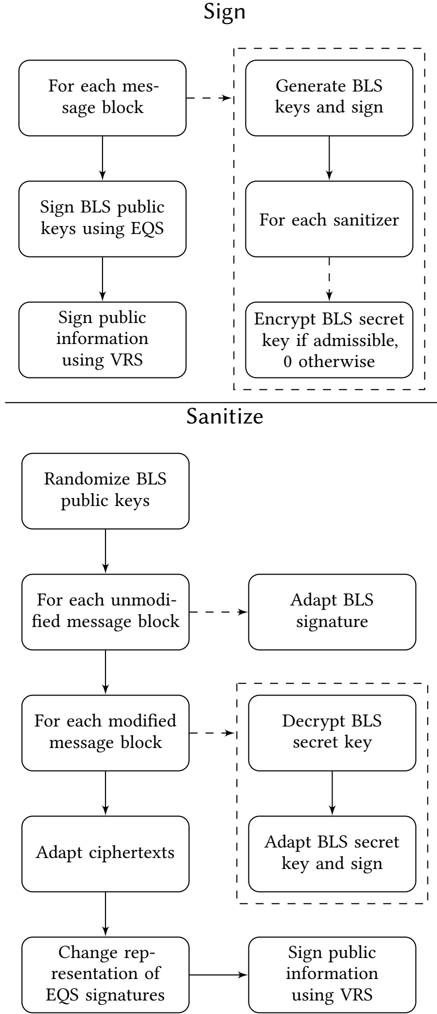

Figure 7: High-level flow chart showing how the Sign **and** Sanitize **algorithms work in** IUT-k-SAN**. A dashed line indicates that the action is done inside a loop and a dashed box groups multiple actions that are done inside the same loop.**

is more significant than the loss of performance on the smartphone side. The results can be found in Figures 10 and 11.

G Efficiency Comparison With the Existing Literature

We compare the efficiency of our constructions to the existing multi-sanitizer schemes [1, 8, 15, 16, 25, 30]. Only [1] calculated the size and complexity of their construction in terms of number of group elements and operations. Other papers 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

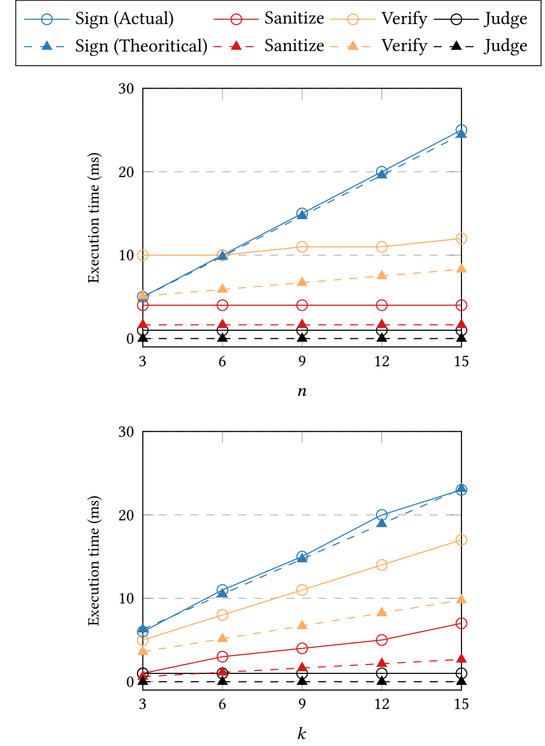

Figure 8: Theoretical and real performance of FSV-k-SAN **on the server. In the first figure, we fix $k = 9$ and test with variable n. In the second one, we fix $n = 9$ and test k.**

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 [8] 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 [15] 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 [16] 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 [16] 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

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.	$\|\sigma\|$	$\|\pi\|$	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]	$O(t)$	$O(1)$	$O(t)\ \mathcal{E}$	$O(t)\ \mathcal{E}$	$O(1)\ \mathcal{E} + O(t)\ \mathcal{P}$	$O(1)\ \mathcal{E}$	$O(1)\ \mathcal{P}$
[8]	$O(1)$	-	$O(1)\ \mathcal{E}$	$O(1)\ \mathcal{E}$	$O(1)\ \mathcal{E}$	-	$O(1)\ \mathcal{E}$
[16]	$O(n)$	-	$O(n)\ \mathcal{E}$	$O(n)\ \mathcal{E}$	$O(n)\ \mathcal{E}$	-	-
[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 $+ O(t)\ \mathcal{E}$	$O(t)\ \mathcal{E} + O(t)\ \mathcal{P}$	$O(1)\ \mathcal{E} + O(1)\ \mathcal{P}$	$O(1)\ \mathcal{E}$	$O(1)\ \mathcal{E}$

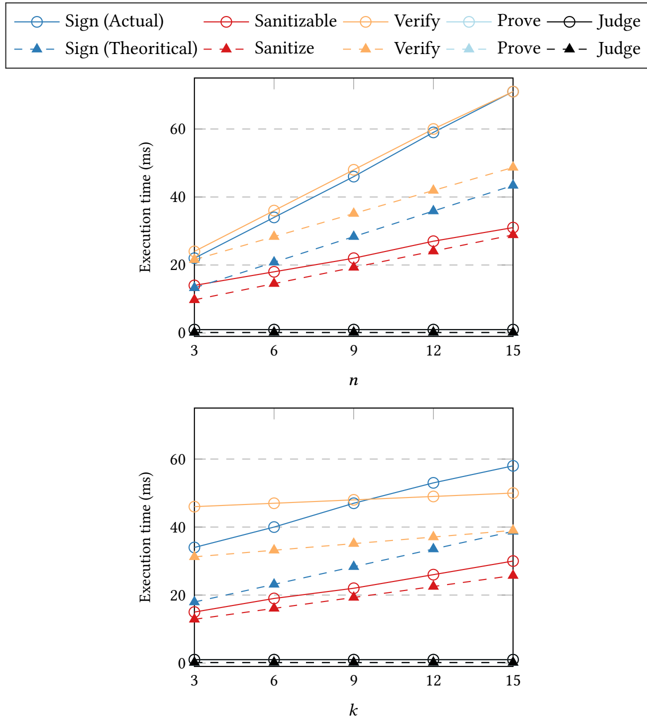

Figure 9: Theoretical and real performance of IUT-k-SAN on the server. In the first figure, we fix $k = 9$ and test with variable n. In the second one, we fix $n = 9$ and test k.

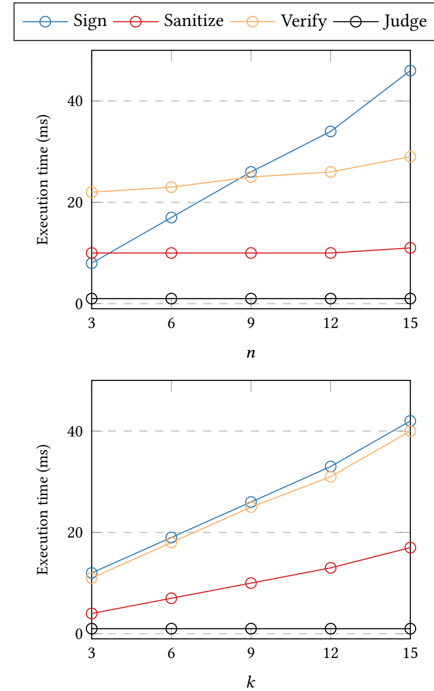

Figure 10: Performance of FSV-k-SAN on the smartphone. In the first figure, we fix $k = 9$ and test with variable n. In the second one, we fix $n = 9$ and test k.

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.

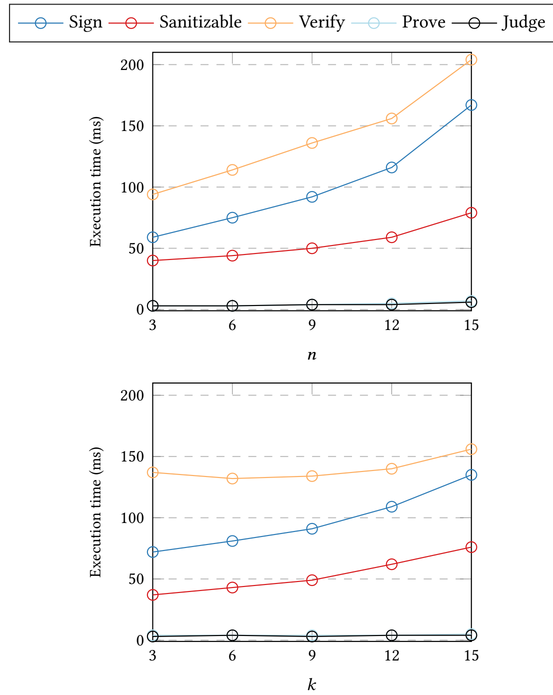

Figure 11: Performance of IUT-k-SAN **on the smartphone. In the first figure, we fix $k = 9$ and test with variable n. In the second one, we fix $n = 9$ and test k.**

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 [8] would require a different signature per set of admissible blocks that are authorized to a certain sanitizer. The constructions of [16, 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 [15] 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.

Received 25 August 2025; revised 30 October 2025; accepted 20 November 2025