Sanitizable Signatures with Several Signers and Sanitizers

Abstract. Sanitizable signatures allow a signer of a message to give one specific receiver, called a sanitizer, the power to modify some designated parts of the signed message. Most of existing constructions consider one single signer giving such a possibility to one single sanitizer. In this paper, we formalize the concept with n signers and m sanitizers, taking into account recent models (for 1 signer and 1 sanitizer) on the subject. We next give a generic construction based on the use of group signatures and on a new cryptographic building block, called a trapdoor or proof, that may be of independent interest.
Keywords. Sanitizable signatures, anonymity, transparency, trapdoor or proof.

1 Introduction

Cryptographic research provides today a large choice of tools to secure our networks and services. Besides authentication and encryption, it exists several ways to lighten or slightly modify the main cryptographic tools. Regarding signature schemes, it is for example possible to blind the identity of the signer (using e.g. group signatures) or to add properties on the resulting message-signature pair.

Among those variants, sanitizable signatures have been introduced in [1] and allow a signer to produce a signature on a document which can be further modified, in a limited and controlled fashion, by a designated "sanitizer". The sanitizer should not interact with the original signer and can only modify the parts of the message that have been stated as modifiable by the signer. As modelled in [8], a sanitizable signature scheme considers one single signer giving the modification power to one single sanitizer[1] and should be *transparent* (only the signer and the sanitizer are able to distinguish an original signature from a sanitized one), *immutable* (it is not possible for the sanitizer to modify non admissible blocks of a signed message), *signer-accountable* (a signer can not force a judge to accuse a sanitizer) and *sanitizer-accountable* (a sanitizer can not force a judge to accuse a signer). The notion of *unlinkability* (infeasibility to identify message-signature pairs from the same source) has later been proposed in [9]. Some extensions [18, 10] have also been described, allowing the signer to better control the modifications the sanitizer can do.

It currently exists several sanitizable signature constructions in the literature [1, 11, 8, 10, 9] and most of them take the case of *one single* signer allowing *one single* sanitizer to sanitize a given message-signature pair. In fact, the only

[1] Even if several sanitizers could exist in the system.

exception, to the best of our knowledge, is the recent work in [9] where the unlinkable sanitizable signature scheme can be extended to the case of "one signer and m sanitizers". But nobody has really taken into account the case of multiple signers and sanitizers in a unique system.

Regarding concrete applications, sanitizable signatures with one signer and one sanitizer may be useful in the context of Digital Right Management [11] (signer of a license vs. modifier of a given license), database applications [1] (commercial vendor vs. database administrator) or medical ones [1] but it can not cover all use cases where sanitizable signatures are useful. In particular, the secure routing application proposed in [1] should use in practice a group of n entities acting as both signers and sanitizers. It can also be used *e.g.* to protect the privacy of customers in a billing system where the service provider does not obtain the identity of the customer and the billing provider does not know the provided service.

Ideas behind our model. In this paper, we propose the first complete model for sanitizable signatures with n signers and m sanitizers. Especially, it totally includes the one of Brzuska *et al.* [8] for 1 signer and 1 sanitizer.

In our setting, a signer can choose several designated sanitizers for a given message. We then redefine accordingly the notions of accountability and immutability and introduce several notions of transparency: the *no-transparency* where anybody can distinguish a signed message from a sanitized one, the *group transparency* where only signers can make such distinction and the *full transparency* where this can only be done by the true signer, the true sanitizer and a designated authority.

As there are several signers and sanitizers, we study the case where the signer (resp. the sanitizer) is anonymous within the group of signers (resp. sanitizers). Similarly to the transparency, we also introduce the concept of *group anonymity* where the anonymity is only respected for people who are not in the group (of signers and sanitizers resp.). To be complete, we also treat the anonymity revocation by some designated authorities and study the notion of traceability and non-frameability from the group signature world [3, 5].

Idea of our construction. Our main (n, m) multi-players sanitizable signature construction is based on the work of Brzuska *et al.* [9], using group signatures [3, 6, 5, 14]. We also give several ideas to improve this initial construction and obtain several different schemes with different security levels. Finally we detail a new solution for a fully transparent and fully anonymous sanitizable signature scheme.

More precisely, we base our new solution on a new cryptographic building block that may be of independent interest: *trapdoor or proof*. Zero-knowledge proofs of knowledge allow one prover to prove to one verifier that she knows some secrets verifying some public relations. An *or proof* enables to prove *e.g.* that the known secret is the discrete logarithm of either y in base g or of z in base h. We introduce the concept and give a practical construction for a *trapdoor or proof* where a given authority can reveal which discrete logarithm is known. Independently, this tool can be used to design electronic voting systems.

The paper is organized as follows. Our model for multi-players sanitizable signature is described in Section 2 and is security in Section 3. Section 4 is dedicated to useful tools and Section 5 to our new *trapdoor or proof* and our main multi-players sanitizable signature scheme is described in Section 6.

2 Multi-Players Sanitizable Signatures

In this section, we introduce the concept of multi-players sanitizable signature scheme, by defining algorithms and keys. Note that the initial model from Brzuska *et al.* [8] is included into ours (see Section 3.4).

2.1 Intuition for the Model

Our aim is to propose a model where one signer (among n) can choose a set of sanitizers (among m) that will be able to sanitize a message-signature pair. Moreover we want to be consistent with the initial model from Brzuska *et al.* [8] where one signer chooses one sanitizer. We thus keep traditional procedures (Sign, Sanitize, Verify, SigOpen and Judge) and security properties: *immutability*, *signer and sanitizer accountabilities* and *transparency*. As we now have a group of signers and sanitizers, we can handle these groups in different ways. One possibility is to consider the case where a signer (resp. sanitizer) can be *anonymous* within her group. In some cases, the other signers (resp. sanitizers) may need to identify the signer (resp. the sanitizer) of a given message: this lead to the concept of *group-anonymity*. Moreover, we consider the case where a signer does not want the other signers to know that she has signed a message. But, in this case, the other signers may be able to distinguish an original signature from a sanitized one, which is related to the (group) *transparency*. We obtain the following cases for anonymity and transparency.

Anonymity	**Transparency**
– *No anonymity*: no.	– *No transparency*: no.
– *Group anonymity*: yes, except for her group.	– *Group transparency*: yes, except for all signers (the sanitizer).
– *Full anonymity*: yes, except for a specific authority[2].	– *Full transparency*: yes, except for the signer (the sanitizer) and a specific authority.

Some of those new properties imply a modification of procedures and some additional oracles.

[2] Regarding traditional anonymous signatures, there is most of the time a possibility to open (or revoke the anonymity of) a signature. Thus, in the full anonymity case, the anonymity can only be revoked by a designated authority.

Full-anonymity: we add a full opener $\mathcal{O}_{\text{FULL}}$ who is able to determine (during the FullOpen procedure) the real producer of a given message-signature pair. Similarly, the origin opener $\mathcal{O}_{\text{ORI}}$ is able to retrieve (during the FindOri algorithm) the signer who is at the origin of a given message-signature pair.
Group transparency: we need to extend the SigOpen procedure such that any signer can carry it out.
Full transparency: we introduce an authority, called the algorithm opener, denoted $\mathcal{O}_{\text{ALG}}$, such that only $\mathcal{O}_{\text{ALG}}$, the true original signer and the true sanitizer (if relevant) are able to prove that one message-signature pair is an original or a sanitization. We thus introduce the AlgOpen procedure which can be executed by $\mathcal{O}_{\text{ALG}}$. Note that the true signer is always able to make such distinction but she is not necessarily able to prove it.

The notion of anonymity and the possibility to revoke this property necessary lead to the notion of *traceability* (the identity of an anonymous signer or an anonymous sanitizer can always be retrieved if needed) and *non-frameability* (the infeasibility to produce a wrong opening).

We do not need to add an unforgeability property since it is implied by the accountability, traceability and non-frameability. In fact, from Proposition 4.2 of [8], we obtain that accountabilities imply unforgeability and the given proof still work in our case. Moreover (cf. Appendix A of [5]), the group signature's unforgeability follows from traceability plus non-frameability.

2.2 General Definition

A multi-players sanitizable signature scheme involves a set of signers, a set of sanitizers, an issuer $\mathcal{I}$ that may be divided into $\mathcal{I}_{\text{SIG}}$ and $\mathcal{I}_{\text{SAN}}$ and an opener $\mathcal{O}$ that may be divided into $\mathcal{O}_{\text{FULL}}$, $\mathcal{O}_{\text{ALG}}$, and $\mathcal{O}_{\text{ORI}}$.

ADM is defined by the signer, given a message m of length ℓ and divided into t blocks, as (i) the length ℓ_i of each block m_i (such that $\ell = \sum_{i=1}^{t} \ell_i$) and (ii) the index of the block which will be modifiable by the sanitizer, i.e. the subset $\mathcal{T}$ of $[1, t]$ such that for all $i \in \mathcal{T}$, m_i is modifiable. By misuse of notation, we say that $i \in \mathsf{ADM}$ if $i \in \mathcal{T}$. If two messages m_0 and m_1 are defined as having the same admissible parts ADM, we note that $\mathsf{ADM}(\mathsf{m}_0, \mathsf{m}_1) = 1$. On input a message m and the variable ADM, the sanitizer define the modifications MOD as the set of all the (i, m'_i) such that she is able to replace the i-th block of m by m'_i. We say that MOD matches ADM if $\forall i \in \mathsf{MOD}$, $i \in \mathsf{ADM}$.

Definition 1 (Multi-players sanitizable signature scheme). *Let λ be a security parameter. A (n, m)-multi-players sanitizable signature scheme Π is composed of the following eleven algorithms.*

Setup(1^λ) *outputs the public key* gpk *of the system, the secret key* $\mathsf{isk} := (\mathsf{isk}_{\text{SIG}}, \mathsf{isk}_{\text{SAN}})$ *of some issuers and, in some cases, an additional opening secret key denoted by* $\mathsf{osk} := (\mathsf{osk}_{\text{FULL}}, \mathsf{osk}_{\text{ALG}}, \mathsf{osk}_{\text{ORI}})$.

SigKG$(1^n, 1^\lambda, \mathsf{isk}_{\text{SIG}})$ *and* SanKG$(1^m, 1^\lambda, \mathsf{isk}_{\text{SAN}})$ *take as input the issuer key* $\mathsf{isk}_{\text{SIG}}$ *(resp.* $\mathsf{isk}_{\text{SAN}}$*), the number n (resp. m) of signers (resp. of sanitizers)* *and λ. They output two vectors of keys $(\boldsymbol{sk}_{\text{SIG}}, \boldsymbol{pk}_{\text{SIG}})$ (resp. $(\boldsymbol{sk}_{\text{SAN}}, \boldsymbol{pk}_{\text{SAN}})$). From now on, the whole public key $(\mathsf{gpk}, \boldsymbol{pk}_{\text{SIG}}, \boldsymbol{pk}_{\text{SAN}})$ is denoted PK.*

$\textsc{Sign}(\mathsf{m}, \boldsymbol{sk}_{\text{SIG}}[i], \widetilde{\boldsymbol{pk}}_{\text{SAN}}, \mathsf{ADM}, \mathsf{PK})$ *enables the signer i to sign a message m for authorized sanitizers $\widetilde{\boldsymbol{pk}}_{\text{SAN}} \subseteq \boldsymbol{pk}_{\text{SAN}}$ according to ADM as defined above. It outputs a signature σ on m. By convention σ contains ADM and $\widetilde{\boldsymbol{pk}}_{\text{SAN}}$. Note that σ also contains the way for authorized sanitizers to sanitize m.*

$\textsc{Sanitize}(\mathsf{m}, \sigma, \boldsymbol{sk}_{\text{SAN}}[j], \mathsf{MOD}, \mathsf{PK})$ *is carried out by the sanitizer j to sanitize a message-signature pair (m, σ). The modifications MOD describe the new message m' as defined above. This algorithm outputs a new signature σ' and the modified message m' or $\perp$ in case of error (for example, j is not able to sanitize this message).*

$\textsc{Verify}(\mathsf{m}, \sigma, \mathsf{PK})$ *allows to verify the signature σ (sanitized or not) on the message m. It outputs 1 if the signature is correct and 0 if it is not.*

$\textsc{FullOpen}(\mathsf{m}, \sigma, \mathsf{osk}_{\text{FULL}}, \mathsf{PK})$ *enables the opener $\mathcal{O}_{\text{FULL}}$ to find the identity of the producer of the given message. It outputs the string* FULL*, an identity I_{FULL} which is either $(\mathsf{sig}, i_{\text{FULL}})$ or $(\mathsf{san}, j_{\text{FULL}})$, and a proof τ_{FULL} of this claim. In case $I_{\text{FULL}} = 0$, it is claiming that no one produced σ.*

$\textsc{AlgOpen}(\mathsf{m}, \sigma, \mathsf{osk}_{\text{ALG}}, \mathsf{PK})$ *enables the opener $\mathcal{O}_{\text{ALG}}$ to find whether the couple (m, σ) is an original or a sanitized couple. It outputs the string* ALG*, next either $I_{\text{ALG}} = \mathsf{sig}$ (original signature) or $I_{\text{ALG}} = \mathsf{san}$ (sanitized signature), and a proof τ_{ALG} of this claim. In case $I_{\text{ALG}} = 0$, the result is that the opener $\mathcal{O}_{\text{ALG}}$ cannot conclude.*

$\textsc{FindOri}(\mathsf{m}, \sigma, \mathsf{osk}_{\text{ORI}}, \mathsf{PK})$ *enables the opener $\mathcal{O}_{\text{ORI}}$ to find the original signer of the given message. It outputs the string* ORI*, the identity $I_{\text{ORI}} = (\mathsf{sig}, i_{\text{ORI}})$ of the original signer and a proof τ_{ORI} of this claim. In case $I_{\text{ORI}} = 0$, it is claiming that no signer is at the origin of σ. Note that i_{ORI} is not necessarily the identity of the actor having produced the signature σ, since this one may have been sanitized after the original signature from i_{ORI}.*

$\textsc{SigOpen}(\mathsf{m}, \sigma, (\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}), \boldsymbol{sk}_{\text{SIG}}[\tilde{i}], \mathsf{PK}, \mathsf{DB})$ *enables the signer $\tilde{i}$ to be convinced, using an entry (*ORI*, I_{ORI}, τ_{ORI}) (with $I_{\text{ORI}} := (\mathsf{sig}, i_{\text{ORI}})$) which could have been produced by the* $\textsc{FindOri}$ *algorithm, that the signer i_{ORI} is the originator of the given message. The signer $\tilde{i}$ may use a set DB of couples (m_k, σ_k) and proves that the given message-signature pair (m, σ) is or is not a sanitized pair. It outputs a triple containing the string* SIG*, either $I_{\text{SIG}} = I_{\text{ORI}}$ if (m, σ) a true signature or $(\mathsf{san}, 0)$ if (m, σ) was sanitized, and a proof τ_{SIG} (including τ_{ORI}). It outputs $I_{\text{SIG}} = 0$ if the signer $\tilde{i}$ can not conclude.*

$\textsc{Judge}(\mathsf{m}, \sigma, \mathsf{gpk}, (s, I_s, \tau_s), \mathsf{PK})$ *is a public algorithm which aims at deciding the origin of a given message-signature pair (m, σ). According to the string $s \in \{$FULL, ALG, ORI, SIG$\}$, it outputs 1 if the predicate guessed in τ_s is exact and 0 otherwise.*

The correctness property (see Appendix A.1) states that all of them should be correct, from the verification to the different opening algorithms.

3 Security Requirements

We now give the security definitions a multi-players sanitizable signature scheme should satisfy. Our work is based on those from [8] and [3].

3.1 Oracles and Adversaries

Oracles. The security properties will be displayed using experiments in which the adversary's attacks are modelled by using call to some oracles. In the following, $\mathcal{CU}$ denotes the set of corrupted users (as a signer or a sanitizer).

- $\mathsf{setup}(\cdot,\cdot,\cdot)$: this oracle corresponds to the generation of the different keys and parameters. It takes as input the parameters $\lambda, n, m \in \mathbb{N}$ and executes the procedures $\textsc{Setup}(\cdot)$, $\textsc{SigKG}(\cdot,\cdot,\cdot)$ and $\textsc{SanKG}(\cdot,\cdot,\cdot)$ and the set PK and $\mathsf{SK} = \{\mathsf{isk}, \mathsf{osk}, \mathbf{sk}_{\text{SIG}}, \mathbf{sk}_{\text{SAN}}\}$ are given on output.
- $\mathsf{corrupt}(\cdot,\cdot,\cdot)$: By calling this oracle, the adversary corrupts a signer or a sanitizer. This oracle takes as input three elements: the first one $a \in \{\mathsf{sig}, \mathsf{san}\}$ says whether the corrupted player is a signer or a sanitizer, the second argument $k \in \mathbb{N}$ gives the identity of the corresponding signer ($k \in [1, n]$) or sanitizer ($k \in [1, m]$) and the third one corresponds to a public key pk. The couple (a, k) is added to the set $\mathcal{CU}$ and the oracle sets $\mathbf{pk}_{\text{SIG}}[k] = \mathsf{pk}$ if $a = \mathsf{sig}$ (or $\mathbf{pk}_{\text{SAN}}[k] = \mathsf{pk}$ if $a = \mathsf{san}$) and $\mathbf{sk}_{\text{SIG}}[k] = \perp$ if $a = \mathsf{sig}$ (or $\mathbf{sk}_{\text{SAN}}[k] = \perp$ if $a = \mathsf{san}$). An adversary having access to no corruption oracle is denoted $\mathcal{A}^{(0)}$, an adversary only having access to $\mathsf{corrupt}(\mathsf{sig},\cdot,\cdot)$ (resp. $\mathsf{corrupt}(\mathsf{san},\cdot,\cdot)$) is denoted $\mathcal{A}^{(\mathsf{si})}$ (resp. $\mathcal{A}^{(\mathsf{sa})}$), while an adversary having access to both is denoted by $\mathcal{A}^{(*)}$.
- $\mathsf{sign}(\cdot,\cdot,\cdot)$, $\mathsf{sanitize}(\cdot,\cdot,\cdot,\cdot)$, $\mathsf{fullopen}(\cdot,\cdot)$, $\mathsf{algopen}(\cdot,\cdot)$, $\mathsf{findorigin}(\cdot,\cdot)$ and finally $\mathsf{sigopen}(\cdot,\ \cdot,\ \cdot,\ \cdot)$: these oracles are related to the procedures given in Definition 1 (without the non necessary public parameters) that can be called by the adversary. The set of queries and answers to and from the sign (resp. the $\mathsf{sanitize}$) oracle is denoted $\mathbf{\Sigma}_{\mathsf{sig}}$ (resp. $\mathbf{\Sigma}_{\mathsf{san}}$) and is composed of elements of the form $(\mathsf{m}_k,\ i_k,\ \widetilde{\mathbf{pk}}_{\text{SAN},k},\ \mathsf{ADM}_k,\ \sigma_k)$ (resp. $(\mathsf{m}_k, \sigma_k, j_k, \mathsf{MOD}_k, m'_k, \sigma'_k)$).

Adversaries. For each property, there are two type of adversary.

1. A generator adversary $\mathcal{A}_{\mathsf{gen}}$ outputs something that will pass some given criteria. In this case, the experiment outputs 1 if all criteria on the adversary's output are verified. For any adversary $\mathcal{A}_{\mathsf{gen}}$ against a property prop and any parameters $\lambda, n, m \in \mathbb{N}$, the success probability of $\mathcal{A}_{\mathsf{gen}}$ is the probability that the related experiment outputs 1. We say that the whole scheme verify the property prop if this success is negligible (as a function of λ, n, m) for any polynomial-time $\mathcal{A}_{\mathsf{gen}}$.
2. A choose-then-guess adversary $\mathcal{A} = (\mathcal{A}_{\mathsf{ch}}, \mathcal{A}_{\mathsf{gu}})$ is divided into two phases: $\mathcal{A}_{\mathsf{ch}}$ for the "choose" phase or $\mathcal{A}_{\mathsf{gu}}$ for the "guess" one. For the experiments, a challenge bit $b \in \{0, 1\}$ is set and for any adversary $\mathcal{A}$ against a property prop and any parameters $\lambda, n, m \in \mathbb{N}$, the advantage of $\mathcal{A}$ is $\Pr\Big[\mathbf{Exp}^{\mathsf{prop}\text{-}1}_{\Pi,\mathcal{A}} =$

$1\Big] - \Pr\Big[\mathbf{Exp}_{\Pi,\mathcal{A}}^{\mathsf{prop}\text{-}0} = 1\Big]$. We next say that the whole scheme verifies the property prop if this advantage is negligible for any polynomial-time $\mathcal{A}$.

3.2 Security Properties

All the formal experiments are given in Figures 1 and 2.

– **Immutability**. The immutability comes from the sanitizable signature scheme part. Informally speaking, it says that it is not possible, for an adversary controlling all the sanitizers, to modify a signed message in a non admissible part. We allow the adversary to corrupt signers, with the condition that the output pair (m^*, σ^*) does not originally come from a corrupted signer.

– **Accountability**. The accountability says that users are responsible for the signatures they produce. The group of sanitizers (resp. signers) cannot convince a judge that a signature has been produced by a signer (resp. a sanitizer) if this is not the case.

– **Sanitizer accountability**. The adversary controls all the sanitizers and outputs a (m^*, σ^*) pair which will be attributed to a signer, while this is not the case. The first possibility for the adversary is to output a valid tuple (ALG, sig, τ_{ALG}) accepted by the judge. The second possibility is to make use of an honest signer i^* of its choice, such that when this signer will execute the SigOpen algorithm, the output will be (SIG, I_{sig}) with I_{sig} referring to an honest signer. Since the adversary is given the ability to corrupt signers, we should be convinced that σ^* neither comes from a corrupted signer nor from the Sign oracle.

– **Signer accountability**. The adversary controls all the signers and outputs a (m^*, σ^*) pair which will be attributed to a sanitizer, while this is not the case. The first possibility for the adversary is to output a valid tuple (ALG, san, τ_{ALG}) accepted by the judge. The second possibility is to produce a proof (SIG, (san, 0), τ_{SIG}), which may be output by the SigOpen procedure, that will be accepted by the judge. Again, since the adversary is given the ability to corrupt sanitizers, we should be convinced that σ^* neither comes from a corrupted sanitizer/signer nor from the Sanitize oracle.

– **Transparency**. The aim of the adversary is here to decide whether a given message-signature is a sanitized one or not. In the *group transparency* case, the adversary is not able to corrupt signers. In the *full transparency* case, she has access to the signer corruption oracle. We notice that the existence of the SigOpen procedure obviously implies that the full transparency can not be reached. Therefore, to design a fully transparent multi-players sanitizable signature scheme, the SigOpen procedure must be restricted to the case $\tilde{i} = i_{\text{ORI}}$.

– **Unlinkability**. The aim of the adversary is here to choose two messages that become identical once sanitized and decide which one has been sanitized given the result of the sanitization procedure. The adversary has access to a left-or-right oracle which executes the sanitization according to a random bit the adversary must guess. We notice that the sanitization procedure may be carried out by different sanitizers, if both are authorized to modify the message.

– **Traceability**. The traceability says that the opening should always conclude. The adversary wins if she is able to output a message-signature pair such that

$\underline{\mathbf{Exp}^{\mathsf{imm}}_{\Pi,\mathcal{A}}(\lambda, n, m)}$:

– $(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{setup}(1^\lambda)$; $(\mathsf{m}^*, \sigma^*) \leftarrow \mathcal{A}^{(*)}_{\mathsf{gen}}(\mathsf{PK}, \mathsf{isk}_{\text{SAN}})$

– $(\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}) \leftarrow \textsc{FindOri}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{ORI}}, \mathsf{PK})$

– if $\big[\textsc{Verify}(\mathsf{m}^*, \sigma^*, \mathrm{PK}) = 1\big]$ and $\big[I_{\text{ORI}} = (\mathsf{sig}, i_{\text{ORI}}) \notin \mathcal{CU}\big]$ and $\big[\forall(m_k,\ i_k,\ \widetilde{\mathsf{pk}}_{\text{SAN},k},$ $\mathsf{ADM}_k,\ \cdot) \in \mathbf{\Sigma}_{\mathsf{sig}}$ s.t. $i_k = i_{\text{ORI}},\ \big(\widetilde{\mathsf{pk}}_{\text{SAN},k} \neq \widetilde{\mathsf{pk}}^*_{\text{SAN}}\big)$ or $\big(\exists \ell \in [1, t_k]$ s.t. $\mathsf{m}_k[\ell] \neq m^*[\ell]$ and $\ell \notin \mathsf{ADM}_k\big)\big]$, then return 1.

$\underline{\mathbf{Exp}^{\mathsf{san\text{-}acc}}_{\Pi,\mathcal{A}}(\lambda, n, m)}$:

– $(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{setup}(1^\lambda)$; $(\mathsf{m}^*, \sigma^*, (I^*_{\text{ALG}}, \tau^*_{\text{ALG}}), I^* := (\mathsf{sig}, i^*)) \leftarrow \mathcal{A}^{(*)}_{\mathsf{gen}}(\mathsf{PK}, \mathsf{osk}, \mathsf{isk}_{\text{SAN}})$;

– $(\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}) \leftarrow \textsc{FindOri}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{ORI}}, \mathsf{PK})$;

– $(\text{SIG}, I_{\text{SIG}}, \tau_{\text{SIG}}) \leftarrow \textsc{SigOpen}(\mathsf{m}^*, \sigma^*, (\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}), \mathsf{sk}_{\text{SIG}}[i^*], \mathsf{PK}, \mathsf{DB})$; where $\mathsf{DB} := \{(m_k, \sigma_k) \mid \forall(m_k, i_k, \cdot, \cdot, \sigma_k) \in \mathbf{\Sigma}_{\mathsf{sig}}$ s.t. $i_k = i_{\text{ORI}}\}$

– if $\big[\textsc{Verify}(\mathsf{m}^*, \sigma^*, \mathsf{PK}) = 1\big]$ and $\big[I_{\text{ORI}} = (\mathsf{sig}, i_{\text{ORI}}) \notin \mathcal{CU}\big]$ and $\big[\forall(m_k, i_k,$ $\widetilde{\mathsf{pk}}_{\text{SAN},k}, \cdot, \sigma_k) \in \mathbf{\Sigma}_{\mathsf{sig}}, (i_k, \mathsf{m}_k, \widetilde{\mathsf{pk}}_{\text{SAN},k}) \neq (i_{\text{ORI}}, \mathsf{m}^*, \widetilde{\mathsf{pk}}^*_{\text{SAN}}))\big]$ and $\big[\big(I^*_{\text{ALG}} = \mathsf{sig} \wedge$ $\textsc{Judge}(\mathsf{m}^*, \sigma^*, (\text{ALG}, I^*_{\text{ALG}}, \tau^*_{\text{ALG}}, \mathsf{PK}) = 1\big)$ or $\big(I^* \notin \mathcal{CU} \wedge I_{\text{ORI}} = I_{\text{SIG}} \wedge \textsc{Judge}(\mathsf{m}^*,$ $\sigma^*, (\text{SIG}, I_{\text{SIG}}, \tau_{\text{SIG}}), \mathsf{PK}) = 1\big)\big]$, then return 1.

$\underline{\mathbf{Exp}^{\mathsf{sig\text{-}acc}}_{\Pi,\mathcal{A}}(\lambda, n, m)}$:

– $(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{setup}(1^\lambda)$; $(\mathsf{m}^*, \sigma^*, (I^*, \tau^*)) \leftarrow \mathcal{A}^{(*)}_{\mathsf{gen}}(\mathsf{PK}, \mathsf{osk}, \mathsf{isk}_{\text{SIG}})$;

– $(\text{FULL}, I_{\text{FULL}}, \tau_{\text{FULL}}) \leftarrow \textsc{FullOpen}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{FULL}}, \mathsf{PK})$;

– if $\big[\textsc{Verify}(\mathsf{m}^*, \underset{\smile}{\sigma^*}, \mathsf{PK}) = 1\big]$ and $\big[I_{\text{FULL}} = (\mathsf{san}, j^*) \notin \mathcal{CU}\big]$ and $\big[\forall(\cdot, \sigma_k, \cdot, \cdot, m'_k, \cdot) \in$ $\mathbf{\Sigma}_{\mathsf{san}}, (\mathsf{pk}_{\text{SAN}}[j^*] \notin \widetilde{\mathsf{pk}}_{\text{SAN},k}$ or $\mathsf{m}'_k \neq \mathsf{m}^*)\big]$ and $\big[\big(I^* = \mathsf{san} \wedge \textsc{Judge}(\mathsf{m}^*, \sigma^*, (\text{ALG}, I^*, \tau^*),$ $\mathsf{PK}) = 1\big)$ or $\big([I^* = (\mathsf{san}, 0) \wedge \textsc{Judge}(\mathsf{m}^*, \sigma^*, (\text{SIG}, I^*, \tau^*), \mathsf{PK}) = 1\big)\big]$, then return 1.

$\underline{\mathbf{Exp}^{\mathsf{tran}\text{-}b}_{\Pi,\mathcal{A}}(\lambda, n, m)\ //\ b \in \{0,1\}}$:

– $(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{setup}(1^\lambda)$; $(\mathsf{m}^*, \mathsf{ADM}^*, \mathsf{MOD}^*, i^*, j^*, \widetilde{\mathsf{pk}}^*_{\text{SAN}}, st) \leftarrow \mathcal{A}^{(*)}_{\mathsf{ch}}(\mathsf{PK})$;

– $\sigma^* \leftarrow \textsc{Sign}(\mathsf{m}^*, \mathsf{sk}_{\text{SIG}}[i^*], \widetilde{\mathsf{pk}}^*_{\text{SAN}}, \mathsf{ADM}^*, \mathsf{PK})$;

– $(\mathsf{m}'^*, \sigma'^*_0) \leftarrow \textsc{Sanitize}(\mathsf{m}^*, \sigma^*, \mathsf{sk}_{\text{SAN}}[j^*], \mathsf{MOD}^*, \mathsf{PK})$;

– if $b = 1$, then $\sigma'^*_1 \leftarrow \textsc{Sign}(\mathsf{m}'^*, \mathsf{sk}_{\text{SIG}}[i^*], \widetilde{\mathsf{pk}}^*_{\text{SAN}}, \mathsf{ADM}^*, \mathsf{PK})$;

– $b^* \leftarrow \mathcal{A}^{(*)}_{\mathsf{gu}}(\mathsf{m}'^*, \sigma'^*_b, st)$

– if $i^* \in \mathcal{CU}$ or $j^* \in \mathcal{CU}$ or $(\mathsf{m}'^*, \sigma'^*_b)$ was queried to $\mathsf{sigopen}$, return $\perp$, else return b^*.

$\underline{\mathbf{Exp}^{\mathsf{unlink}\text{-}b}_{\Pi,\mathcal{A}}(\lambda, n, m)\ //\ b \in \{0,1\}}$:

– $(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{setup}(1^\lambda)$; $(\mathsf{m}^*_0, \mathsf{m}^*_1, \sigma^*_0, \sigma^*_1, \mathsf{MOD}^*_0, \mathsf{MOD}^*_1, j^*_0, j^*_1, st) \leftarrow \mathcal{A}^{(*)}_{\mathsf{ch}}(\mathsf{PK})$;

– $(\mathsf{m}'^*, \sigma'^*) \leftarrow \textsc{Sanitize}(\mathsf{m}^*_b, \sigma^*_b, \mathsf{sk}_{\text{SAN}}[j^*_b], \mathsf{MOD}^*_b, \mathsf{PK})$;

– $b^* \leftarrow \mathcal{A}^{(*)}_{\mathsf{gu}}(\mathsf{m}'^*, \sigma'^*_b, st)$

– if $I_{\text{ORI}} = 0$ or ($I_{\text{ORI}} = (\mathsf{sig}, i_{\text{ORI}})$ and $i_{\text{ORI}} \in \mathcal{CU}$) or $\textsc{Judge}(m'^*, \sigma'^*_b, (\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}),$ $\mathsf{PK}) = 0$ or $j_0 \in \mathcal{CU}$ or $j_1 \in \mathcal{CU}$ or $(\mathsf{m}'^*, \sigma'^*)$ was queried to $\mathsf{sigopen}$, return $\perp$, else return b^*.

Fig. 1. Security experiments coming from sanitizable signatures

the opener $(\mathcal{O}_{\text{FULL}}, \mathcal{O}_{\text{ORI}})$ outputs $\perp$ or is unable to produce a correct proof τ of

$\underline{\mathbf{Exp}^{\mathsf{trac}}_{\Pi,\mathcal{A}}(\lambda, n, m)}$:

– $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{setup}(1^\lambda)$; $(\mathsf{m}^*, \sigma^*) \leftarrow \mathcal{A}^{(*)}_{\mathsf{gen}}(\mathsf{PK}, \mathsf{osk})$;
– $(\text{FULL}, I_{\text{FULL}}, \tau_{\text{FULL}}) \leftarrow \text{FULLOPEN}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{FULL}}, \mathsf{PK})$;
– $(\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}) \leftarrow \text{FINDORI}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{ORI}}, \mathsf{PK})$;
– if $\big[\text{VERIFY}(\mathsf{m}^*, \sigma^*, \mathsf{PK}) = 1\big]$ and $\big[\text{JUDGE}(\mathsf{m}^*, \sigma^*, (s, I_s, \tau_s), \mathsf{PK}) = 0$ or $I_{\text{FULL}} = 0$ or $I_{\text{ORI}} = 0\big]$, then return 1.

$\underline{\mathbf{Exp}^{\mathsf{san\text{-}ano\text{-}}b}_{\Pi,\mathcal{A}}(\lambda, n, m)\ //\ b \in \{0,1\}}$:

– $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{setup}(1^\lambda)$; $(j_0^*, j_1^*, \mathsf{m}^*, \sigma^*, \mathsf{MOD}^*, st^*) \leftarrow \mathcal{A}^{(\mathsf{si})}_{\mathsf{ch}}(\mathsf{PK}, \mathsf{isk}_{\text{SIG}})$ $[\mathcal{A}^{(*)}_{\mathsf{ch}}$ full-anon$]$
– $(\mathsf{m}'^*, \sigma'^*) \leftarrow \text{SANITIZE}(\mathsf{m}^*, \sigma^*, \mathbf{sk}_{\text{SAN}}[j_b^*], \mathsf{MOD}^*, \mathsf{PK})$;
– $b^* \leftarrow \mathcal{A}^{(\mathsf{si})}_{\mathsf{gu}}(\mathsf{m}'^*, \sigma'^*, st)$ $[\mathcal{A}^{(*)}_{\mathsf{gu}}$ full-anon$]$;
– if $\text{VERIFY}(\mathsf{m}^*, \sigma^*, \mathsf{PK}) = 0$ or $j_0^* \in \mathcal{CU}$ or $j_1^* \in \mathcal{CU}$ or $(\mathsf{m}'^*, \sigma'^*)$ was queried to $\mathsf{fullopen}$, return $\perp$, else return b^*.

$\underline{\mathbf{Exp}^{\mathsf{sig\text{-}ano\text{-}}b}_{\Pi,\mathcal{A}}(\lambda, n, m)\ //\ b \in \{0,1\}}$:

– $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{setup}(1^\lambda)$; $(i_0^*, i_1^*, \mathsf{m}^*, \widetilde{\mathbf{pk}}^*_{\text{SAN}}, \mathsf{ADM}^*, st) \leftarrow \mathcal{A}^{(\mathsf{sa})}_{\mathsf{ch}}(\mathsf{PK}, \mathsf{isk}_{\text{SAN}})$ $[\mathcal{A}^{(*)}_{\mathsf{ch}}]$;
– $\sigma^* \leftarrow \text{SIGN}(\mathsf{m}^*, \mathbf{sk}_{\text{SIG}}[i_b^*], \widetilde{\mathbf{pk}}^*_{\text{SAN}}, \mathsf{ADM}^*, \mathsf{PK})$;
– $b^* \leftarrow \mathcal{A}^{(\mathsf{sa})}_{\mathsf{gu}}(\mathsf{m}^*, \sigma^*, st)$ $[\mathcal{A}^{(*)}_{\mathsf{gu}}$ full-anon$]$;
– if $i_0^* \in \mathcal{CU}$ or $i_1^* \in \mathcal{CU}$ or (m^*, σ^*) was queried to $\mathsf{fullopen}$, return $\perp$, else return b^*.

$\underline{\mathbf{Exp}^{\mathsf{nf}}_{\Pi,\mathcal{A}}(\lambda, n, m)}$:

– $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{setup}(1^\lambda)$; $(\mathsf{m}^*, \sigma^*, i^*, \tau^*) \leftarrow \mathcal{A}^{(*)}_{\mathsf{gen}}(\mathsf{PK}, \mathsf{isk}, \mathsf{osk})$;
– If $\big[\text{VERIFY}(\mathsf{m}^*, \sigma^*, \mathsf{PK}) = 1\big]$ and $\big[I^* \notin \mathcal{CU}\big]$ and $\big[(I^* = (\mathsf{sig}, i^*) \wedge \forall(\mathsf{m}_k, i_k, \cdot, \cdot, \cdot) \in \mathbf{\Sigma}_{\mathsf{sig}}, (i_k, \mathsf{m}_k) \neq (i^*, \mathsf{m}^*))$ or $(I^* = (\mathsf{san}, j^*) \wedge \forall(\cdot, \cdot, j_k, \cdot, \mathsf{m}'_k, \cdot) \in \mathbf{\Sigma}_{\mathsf{san}}, (j_k, \mathsf{m}'_k) \neq (j^*, \mathsf{m}^*))\big]$ and $\big[\exists s \in \{\text{FULL}, \text{ORI}\} : \text{JUDGE}(\mathsf{m}^*, \sigma^*, (s, I^*, \tau^*)) = 1\big]$, then return 1.

Fig. 2. Security experiments coming from anonymous signatures

its claim. The power of the adversary is important since she controls all signers and all sanitizers, and is given (but does not controls) the opening keys.
– **Anonymity**. The anonymity property is divided into several ones, depending of the studied user (sanitizer or signer) and the power of the adversary.

– **Sanitizer anonymity**. The adversary here controls all the signers, chooses two sanitizers (j_0^*, j_1^*), a pair (m^*, σ^*) and some MOD^* of her choice. Then the j_b^*-th sanitizer sanitizes the signature (for a uniformly chosen bit b) and the adversary aims at guessing b. In the *group anonymity* case, the adversary is not able to corrupt sanitizers. This is the case when other sanitizers are able to know who has sanitized a given message while it is not possible for other "outside" users. We argue that this notion may be useful in some practical applications. In the *full anonymity* case, she has access to the sanitizer corruption oracle. Note that the "no-" and "group-" anonymity can only be defined if the signer is also viewed as a sanitizer (the contrary not being true) because of the transparency property.

– **Signer anonymity**. The adversary now controls all the sanitizers and aims at distinguish between two signers (i_0^*, i_1^*) of her choice, which one has signed a message m^* according to a chosen ADM^*. We next make the same division as for the sanitizer anonymity part. In the *group anonymity* case, the adversary is not able to corrupt signers. In the *full anonymity* case, she has access to the signer corruption oracle.

– **Non-Frameability**. The non-frameability property argues that it is not possible for an adversary, even being the openers and the group manager, to falsely accuse an honest user (signer or sanitizer) from having produced a valid signature. This property is different from the accountability ones since the aim of the adversary is to produce a judge-accepted proof that an honest user (being a signer or a sanitizer) will be accused of having produced this signature. As a consequence, this takes into account the case where some corrupted signers (resp. sanitizers) try to accuse an honest signer (resp. sanitizer). Moreover, we study the case of a false accusation during the FullOpen and FindOri procedures. The adversary does not control all users but can corrupt them, as it wants. It finally outputs a valid (m^*, σ^*) pair and a (i^*, τ^*) pair which could have been output by the FullOpen (resp. FindOri) procedure. She wins if the judge outputs that i^* has truly produced σ^*, while this is not the case.

3.3 Relation Among Properties

Even if relations exist between security properties, no implication remains. This is less obvious in the relationship between non frameability and accountability but : (i) an adversary against accountability and using the AlgOpen procedure to win the experiment is unable to win against the non frameability experiment ; (ii) an adversary against the non-frameability is stronger (as he controls all the issuing keys isk) than an adversary against the san- (resp. sig-)accountability (who only controls $\mathsf{isk}_{\text{SAN}}$ (resp. $\mathsf{isk}_{\text{SIG}}$)).

3.4 Suitability with Simple Sanitizable Signature Schemes

In [8], Brzuska *et al.* describe a complete model for sanitizable signatures where there is *one* signer who generates signatures that are sanitizable by *one* sanitizer. Ideally, our model, which consider n signers and m sanitizers, should include the Brzuska *et al.* model. Let us thus consider our above model with $n = m = 1$. Obviously, the FindOri procedure is not relevant any more. The AlgOpen and FullOpen collapse and may have an interest in the $(1, 1)$ case but can also be withdrawn to be in accordance with the initial model. Finally, the SigOpen (see the Proof procedure in [8]) should be adapted with $\tilde{i} = i_{\text{ORI}}$. The other procedures remain unchanged.

As there are only one signer and one possible sanitizer, the anonymity property is necessarily the no-anonymity one, for both the signer and the sanitizer case. Next, the non-frameability and the traceability properties have no sense and are naturally withdrawn. Regarding the transparency property, it remains the full one and the no transparency. The initial definition for sanitizable signature [8], with some kind of full transparency, is thus included into our definition. The treatment of the immutability, accountabilities and unlinkability [9] are next easy to solve, removing from the corresponding experiments, given in Figures 1 and 2, the conditions to accept the outputs of the adversary that are no more relevant.

3.5 Adding Extensions

In [10], Canard and Jambert have proposed an extended model in the one signer and one sanitizer case. They specify the following four extensions :

– $\mathsf{LimitSet}$ allows the signer to force a block to be modified only in a set of allowed sub-messages.
– $\mathsf{EnforceModif}$ permits the signer to force the sanitizer to modify in the same manner several admissible blocks.
– $\mathsf{LimitNbModif}$ limits the number of blocks a sanitizer can modify.
– $\mathsf{LimitNbSanit}$ proposes to limit the number of new messages generated by a sanitizer.

Our general model may be extended to consider those extensions in the same manner, thus with the addition of the two algorithms designed for extensions and the two extended security properties defined in [10].

As $\mathsf{LimitSet}$ and $\mathsf{EnforceModif}$ consider only the message itself, it does not interact with the possibility of multiple signers and sanitizers. Both extensions are thus straightforward. But $\mathsf{LimitNbModif}$ and $\mathsf{LimitNbSanit}$ may be defined uniquely for all sanitizers or may be different from one sanitizer to another. Therefore, the limits defined for those extensions may be either defined for all sanitizers or by sanitizer. As the definitions proposed in [10] for those extensions rely on a system of fraud detection, both versions (all sanitizer/by sanitizer) may be easily considered in an extended version of our model.

4 Primitives

Before giving a construction, let us begin by describing some cryptographic primitives we will use.

Digital signature schemes. We will need a standard signature scheme $\mathcal{S} = (\textsc{KGn}, \textsc{Sign}, \textsc{Verif})$ specified by algorithms for key generation, signing and verifying. It should satisfy the standard notion of unforgeability under chosen message attack [17]. In a nutshell, the adversary is given the public key and can interact with a signing oracle. Finally, the adversary outputs an attempted forgery (m, σ) and wins if σ is valid, and m was never queried to the signing oracle. We denote by $\mathbf{Succ}^{\mathsf{unf}}_{\mathcal{S},\mathcal{A}}(\lambda)$ the success probability of the adversary $\mathcal{A}$ against $\mathcal{S}$, with security λ.

Pseudo-random functions. Let $\mathcal{PRF} = (\textsc{FKGn}, \mathrm{PRF})$ be a pseudo-random function, which is defined by the generation algorithm and the pseudo-random function itself. An adversary $\mathcal{A}$ against such scheme is given access to a random function oracle and outputs a value x_0. After that, a bit b is secretly and randomly chosen. If $b = 0$, the adversary receives in return the output of the $\mathcal{PRF}$

on x_0. If $b = 1$, the adversary receives in return a random value. The adversary finally outputs a bit b'. The advantage $\mathbf{Adv}^{\mathsf{prf}}_{\mathcal{PRF},\mathcal{A}}(\lambda)$ of $\mathcal{A}$ is the difference between $1/2$ and the probability that $b' = b$.

Group Signatures. We now formally describe the concept of group signature, as described by Bellare, Shi and Zhang in [5]. In fact, we will need two different types of group signature schemes. First, a BSZ type group signature scheme [5] and second, a similar concept where we do not want an interactive join protocol between the group manager and a group member. This is an hybrid model between the BMW model [3] for static groups and the BSZ [5] one for dynamic groups. We need a non-frameability property to ensure the accountability, since the signer needs to produce a signature without the presence of the sanitizers, as in [9].

A group signature scheme $\mathcal{GS}$ is composed of an issuer, an opener and members, or users, and is given by a tuple (GKGn,UKGn, Join, [NI-Join, GSKGn,] GSign, GVf, Open, Judge) described as follows. The join protocol is denoted NI-Join in case of a non-interactive procedure and it is next necessary for each user to next execute the GSKGn procedure. In Join is interactive, the latter is not necessary. Let λ be a security parameter.

GKGn is a probabilistic algorithm which on input 1^λ outputs the key pair $(\mathsf{ik}, \mathsf{gpk}_i)$ of the issuer (sub-procedure called IGKGn), the key pair $(\mathsf{ok}, \mathsf{gpk}_o)$ of the opener (sub-procedure called OGKGn) and the group public key $\mathsf{gpk} = (\mathsf{gpk}_i, \mathsf{gpk}_o)$.

UKGn is a probabilistic algorithm executed by each user i and which on input 1^λ outputs her key pair $(\mathbf{upk}[i], \mathbf{usk}[i])$.

Join is an interactive protocol between the issuer taking on input ik and $\mathbf{upk}[i]$ and user i taking on input $\mathbf{usk}[i]$. The issuer makes a new entry $\mathbf{reg}[i]$ in its registration table $\mathbf{reg}$. The new group member i obtains a private signing key denoted $\mathbf{msk}[i]$.

[NI-Join is an algorithm executed by the issuer taking on input ik and $\mathbf{mpk} \subseteq \mathbf{upk}$. The issuer outputs its registration table $\mathbf{reg}$.

GSKGn is an algorithm executed by a group member i that on input $\mathbf{usk}[i]$ and $\mathbf{reg}[i]$ outputs a private signing key denoted $\mathbf{msk}[i]$.]

GSign is a probabilistic algorithm that takes on input a message m and a private signing key $\mathbf{msk}[i]$ and outputs a group signature σ on m.

GVerif is a deterministic algorithm that on input a message m, a group signature σ and the group public key gpk outputs 1 if the signature is valid, and 0 otherwise.

Open is an algorithm which on input a message m, a group signature σ and the opener key ok outputs (in a deterministic way) an integer $i \geq 0$ and (in a probabilistic way) a proof τ that i has produced the signature σ on m. If $i = 0$, then no group member produced σ.

Judge is a deterministic algorithm taking on input a message m, a group signature σ, an integer i, the public key $\mathbf{upk}[i]$ of the entity with identity i and a proof-string τ. It outputs 1 if the proof τ is valid and 0 otherwise.

– **Anonymity.** The anonymity property says that the adversary, given signatures produced by a user (among two of his choice) is not able to guess which users provided the signatures. During the related experiment, $\mathcal{A}$ is given access to ik, can corrupt user, obtain their keys, ask for the opening of group signatures and has access to a challenge oracle which takes as input two non-corrupted member i_0 and i_1 and a message m and outputs the group signature of user i_b, for a bit b set by the experiment. Eventually, $\mathcal{A}$ outputs a bit b'. Next, the advantage $\mathbf{Adv}^{\mathsf{ano}}_{\mathcal{GS},\mathcal{A}}(\lambda)$ of $\mathcal{A}$ is the difference between $1/2$ and the probability that $b' = b$.

– **Traceability.** This property says that the adversary is not able to output a valid group signature such that the opening and judge procedures do not occur properly. During the related experiment, $\mathcal{A}$ is given access to ok and outputs (m, σ) which is accepted by the experiment if σ is a valid group signature on m, and either the opening procedure outputs $i = 0$ or the Judge procedure cannot succeed. The success probability $\mathbf{Succ}^{\mathsf{trac}}_{\mathcal{GS},\mathcal{A}}(\lambda)$ the adversary $\mathcal{A}$ is next the probability that the experiment accepts.

– **Non-frameability.** This property says that an adversary $\mathcal{A}$ is not able to falsely accuse an honest user from having produced a valid group signature. During the related experiment, $\mathcal{A}$ is given access to $(\mathsf{ik}, \mathsf{ok})$ and outputs $(\mathsf{m}, \sigma, i, \tau)$ which is accepted by the experiment if σ is a valid group signature on m, i is not corrupted (and her keys are unknown) and the judge accepts the proof τ that i has produced σ while this is not the case. The success probability $\mathbf{Succ}^{\mathsf{nf}}_{\mathcal{GS},\mathcal{A}}(\lambda)$ the adversary $\mathcal{A}$ is next the probability that the experiment accepts.

5 A New Tool: Trapdoor "or" Proof

Let $\textsc{Rel} = \{(x, w)\}$ be a binary relation. We first consider the protocol, corresponding to a proof of knowledge for $\textsc{Rel}$, which is played by a prover, taking on input x and a witness w, and a verifier taking on input x. In fact, following [13, 20], we consider a set $\mathcal{X} = (x_1, \cdots, x_\ell)$ and a proof of knowledge of the "or" statement where both the prover and the verifier take the common input $\mathcal{X}$, while the prover is also given a private input w_i such that $\exists x_i \in \mathcal{X}$ such that$(x_i, w_i) \in \textsc{Rel}$. Additionally to the witness itself, the verifier should not be able to obtain the index i related to x_i.

In our construction, a designated entity should be able to know which index i is really used by the witness of a user to verify $\textsc{Rel}$, while it is still infeasible for every other actors. To the best of our knowledge, this notion of *trapdoor or proof* does not exist in the literature. However, it can be very useful, as we will see later for our main construction of an (n, m)-sanitizable signature scheme, but also *e.g.* for e-voting where the result of the vote (candidate A "or" candidate B) should not be known, except by authorized scrutineers.

DEFINITIONS. In the following, the above or proof is next denoted

$$\textsc{TPok}(w_i : \exists i \in [1, \ell] | (x_i, w_i) \in \textsc{Rel})$$

and the whole system, including the key generation TKGN for the trap, and the "opening" procedure TOPEN, is denoted $\mathcal{TOP} = (\text{TKGN}, \text{TPOK}, \text{TOPEN})$.

As usual, such a proof of knowledge should verify the completeness (a valid prover knowing one such w_i is accepted with overwhelming probability), the soundness (a false prover who does not know any such w_i should be rejected with overwhelming probability) and the honest-verifier zero-knowledge properties (the proof does not reveal any information about the witness).

CIPHER COMMUTING RELATIONS. In the following, we will describe a way to generically design a trapdoor or proof for any relation REL. For this purpose, we need to commute the relation and the encryption procedure of a public key encryption scheme and we thus need to restrict the relations where such commuting operation is possible, which gives us the following definition.

Definition 2 (Cipher commuting relation). *Let λ be a security parameter. Let $\mathcal{E} = (\text{EKG}, \text{ENC}, \text{DEC})$ be a secure probabilistic encryption scheme. Let* REL *be a binary relation. We say that* REL *is a* cipher commuting relation *if for all x, w, for all $(\mathsf{epk}, \mathsf{esk}) \longleftarrow \text{EKG}(1^\lambda)$,*

$$(x, w) \in \text{REL} \Longleftrightarrow (\text{ENC}(x, \mathsf{epk}), w) \in \text{REL}.$$

OUR GENERIC CONSTRUCTION. Let λ be a security parameter, $\mathcal{E} = (\text{EKG}, \text{ENC}, \text{DEC})$ be a secure probabilistic encryption scheme and REL be a cipher commuting relation. We want to design the proof $\text{TPOK}(w_i : \exists i \in [1, \ell] | (x_i, w_i) \in \text{REL})$ where the prover knows w_i such that $(x_i, w_i) \in \text{REL}$.

In a nutshell, we encrypt x_i and use a traditional or proof that the encrypted value is one element related to REL, without revealing which one. We next use the cipher commuting property of REL to prove that the knowledge of a witness which verifies REL with the cipher c_i related to x_i.

Let us first consider that the trap has been generated by executing $(\mathsf{epk}, \mathsf{esk}) \leftarrow \text{EKG}(1^\lambda)$. The proof next works as follows.

1. Computes $c_i = \text{ENC}(x_i, \mathsf{epk})$.
2. Generates the standard honest-verifier zero-knowledge proof with both relations:
   (a) $\text{POK}(x_i : \exists i \in [1, \ell] | c_i = \text{ENC}(x_i, \mathsf{epk}))$ and
   (b) $\text{POK}(w_i : (\text{ENC}(x_i, \mathsf{epk}), w_i) \in \text{REL})$.

As they are connected with an "and", these relations can be composed together, using standard techniques [12].

Finally, the owner of esk can easily decrypt c_i to retrieve x_i. In Appendix C, we give a concrete instantiation based on Schnorr and related to the knowledge of a discrete logarithm.

6 Full Transparent and Fully Anonymous Multi-Players Sanitizable Signature

We now describe our *fully transparent* and *fully anonymous* sanitizable signature scheme for several signers and sanitizers. Our scheme is composed of openers

$\mathcal{O}_{\text{FULL}}$, $\mathcal{O}_{\text{ALG}}$, $\mathcal{O}_{\text{ORI}}$, a group manager $\mathcal{GM}$ for a group signature scheme and a certification authority denoted $\mathcal{CA}$.

6.1 Generation Phases

Let μ be a security parameter. Let $\mathcal{GS}_1$ (resp. $\mathcal{GS}_2$) be an interactive-join (resp. non-interactive-join) group signature scheme, let $\mathcal{S}$ be a standard signature scheme, let $\mathcal{TOP}$ be a trapdoor or proof system (see Section 5) and let $\mathcal{PRF}$ be a pseudo-random function.

SETUP PHASE. The certification authority $\mathcal{CA}$ executes twice the key generation $\mathcal{S}$.KGN for the standard signature scheme $\mathcal{S}$. This gives it two different secret keys denoted $\mathsf{cask_{si}}$ and $\mathsf{cask_{sa}}$ and two related public keys $\mathsf{capk_{si}}$ and $\mathsf{capk_{sa}}$. The group manager $\mathcal{GM}$ executes $\mathcal{GS}_1$.IGKGN, which gives isk and gpk_i. The opener $\mathcal{O}_{\text{FULL}}$ executes $\mathcal{GS}_2$.OGKGN, which gives $\mathsf{osk}_{\text{FULL}}$ and $\mathsf{gpk}_{\text{FULL}}$. The opener $\mathcal{O}_{\text{ALG}}$ execute the $\mathcal{TOP}$.TKGN algorithm, which gives it $\mathsf{osk}_{\text{ALG}}$ and $\mathsf{gpk}_{\text{ALG}}$. The opener $\mathcal{O}_{\text{ORI}}$ executes $\mathcal{GS}_1$.OGKGN, which gives $\mathsf{osk}_{\text{ORI}}$ and gpk_o. In the following, we denote $\mathsf{gpk_{si}} = (\mathsf{gpk}_i, \mathsf{gpk}_o)$.

We finally have $\mathsf{isk}_{\text{SIG}} = (\mathsf{isk}, \mathsf{cask_{si}})$, $\mathsf{isk}_{\text{SAN}} = (\mathsf{cask_{sa}})$, $\mathsf{gpk} = (\mathsf{capk_{si}}, \mathsf{capk_{sa}}, \mathsf{gpk_{si}}, \mathsf{gpk}_{\text{FULL}}, \mathsf{gpk}_{\text{ALG}})$.

SIGNER KEY GENERATION. Each signer i executes the following actions. She executes the JOIN interactive protocol with $\mathcal{GM}$ and obtains her private signing key denoted $\mathbf{msk}[i]$. Next, she executes the key generation for the pseudo-random function: $\mathcal{PRF}$.FKGEN(1^μ) and obtain $\mathsf{uk}[i]$. After that, she uses the user key generation $\mathcal{GS}_2$.UKGN for $\mathcal{GS}_2$ to obtain both $\mathbf{upk}_{si}[i]$ and $\mathbf{usk}_{si}[i]$. Finally, she sends $\mathbf{upk}_{si}[i]$ to $\mathcal{CA}$ together with a non-interactive proof of knowledge of the related $\mathbf{usk}_{si}[i]$. $\mathcal{CA}$ next executes $\mathcal{S}$.SIGN$(\mathbf{upk}_{si}[i], \mathsf{cask_{si}})$ which corresponds to $\mathbf{uc}_{si}[i]$.

SANITIZER KEY GENERATION. Finally, each sanitizer executes the following actions. She executes $\mathcal{GS}_2$.UKGN and obtain both $\mathbf{upk}_{sa}[j]$ and $\mathbf{usk}_{sa}[j]$. She next sends $\mathbf{upk}_{sa}[j]$ to $\mathcal{CA}$ and a non-interactive proof of knowledge of the related $\mathbf{usk}_{sa}[j]$. $\mathcal{CA}$ next executes $\mathcal{S}$.SIGN$(\mathbf{upk}_{sa}[j], \mathsf{cask_{sa}})$ which corresponds to $\mathbf{uc}_{sa}[j]$.

6.2 Signature Procedure

During the signing procedure, the signer first generates the keys of a new group signature scheme (for herself and the chosen sanitizers). She next produces two different group signatures. The first one is a group signature as a member of the group of signers and the second signature as a member of the new formed group. Let us consider the i-th signer, $i \in [1, n]$ and let m be the message she wants to sign. As described in Section 2.1, this message is divided into t parts. Following ADM given on input, let $\mathsf{m}_{\mathtt{FIX}}$ be the part of m which will not be sanitizable by the sanitizers. The SIGN procedure is described as follows.

CHOICE OF SANITIZERS. She chooses a subset $J \subseteq [1, m]$ of sanitizers who will be able to sanitize the message-signature. Let $\widetilde{\mathbf{pk}}_{\text{SAN}}$ be the set of sanitizer's public keys in J.

Generation of a group. The signer next creates a group including herself and the authorized sanitizers she has just chosen. For this purpose, she uses the group signature scheme $\mathcal{GS}_2$ with a non-interactive join. More precisely, she computes $rd = \mathsf{PRF}(\mathsf{uk}[i], id_{\mathsf{m}})$ where $id_{\mathsf{m}} = \mathsf{m}_{\mathtt{FIX}} \| \mathsf{ADM}$ is the identifier of the initial message.

She next carries out the key generation algorithm IGKGn of the group signature scheme $\mathcal{GS}_2$, using rd as a random (see also [9]). It gives $\mathsf{isk}[i, id_{\mathsf{m}}]$ and $\mathsf{gpk}_i[i, id_{\mathsf{m}}]$ (and next $\mathsf{gpk}[i, id_{\mathsf{m}}] = (\mathsf{gpk}_i[i, id_{\mathsf{m}}], \mathsf{gpk}_{\text{FULL}}))$. She next executes the non-interactive join procedure NI-Join of $\mathcal{GS}_2$ to generate the private signing key for all group members, using rd as a random. For the signer, we denote it $\mathbf{reg}[i, id_{\mathsf{m}}]$. For the sanitizers, we denote it $\mathbf{reg}_{sa}[j, id_{\mathsf{m}}]$ for all $j \in J$.

She finally generates her own membership secret key for this group by executing the GSKGn procedure on input $\mathbf{upk}_{si}[i]$ and $\mathbf{reg}[i, id_{\mathsf{m}}]$ to compute $\mathbf{msk}[i, id_{\mathsf{m}}]$.

Group signatures generation. The next step is to compute the message $\mathsf{m}_{\text{SIG}} = id_{\mathsf{m}} \| \widetilde{\mathbf{pk}}_{\text{SAN}} \| \mathbf{reg}_{sa}$. Next, the two following group signatures are generated: $\sigma_{\mathtt{fix}} = \mathcal{GS}_1.\text{GSign}(\mathbf{msk}[i], \mathsf{m}_{\text{SIG}})$ on behalf of the group of signers and $\sigma_{\mathtt{full}} = \mathcal{GS}_2.\text{GSign}(\mathbf{msk}[i, id_{\mathsf{m}}], \mathsf{m})$ for the new formed group.

Proof of validity. The signer finally proves that the above is correctly done and that she is either a signer or a sanitizer. For this purpose, she produces the non-interactive zero-knowledge proofs of knowledge π:

- PoK($\mathbf{msk}[i, id_{\mathsf{m}}]$, $\mathbf{upk}_{si}[i]$, $\mathbf{reg}[i, id_{\mathsf{m}}]$) : $\mathbf{msk}[i, id_{\mathsf{m}}] = \mathcal{GS}_2$.GSKGn($\mathbf{upk}_{si}[i]$, $\mathbf{reg}[i, id_{\mathsf{m}}]$)); and
- PoK($\mathbf{msk}[i, id_{\mathsf{m}}]$: $\sigma_{\mathtt{full}} = \mathcal{GS}_2$.GSign($\mathbf{msk}[i, id_{\mathsf{m}}], \mathsf{m}$)); and
- TPOK($\mathbf{upk}_{si}[i]$, $\mathbf{uc}_{si}[i]$: $\exists \mathbf{sk} \in \{\mathsf{cask}_{\mathsf{si}}, \mathsf{cask}_{\mathsf{sa}}\} | \mathbf{uc}_{si}[i] = \mathcal{S}$.Sign($\mathbf{upk}_{si}[i]$, $\mathbf{sk}$)).

Again, as the proofs are connected with an "and" (which is the case for the trapdoor or proof), these relations can be composed together [12].

The resulting signature is finally $\sigma = (\pi, \sigma_{\mathtt{FIX}}, \sigma_{\mathtt{FULL}}, \mathsf{ADM}, \widetilde{\mathbf{pk}}_{\text{SAN}}, \mathsf{w}_i, \mathbf{reg}_{sa})$ where $\mathbf{reg}_{sa}$ allows to avoid the necessity for each sanitizer to be on-line to obtain their group member keys.

6.3 Sanitization Procedure

The sanitization algorithm consists, for the sanitizer, in creating a new $\sigma'_{\mathtt{FULL}}$, according to the modification she is doing to the message and her own keys, and to produce the corresponding modified proof π'. Let us consider the j-th sanitizer, $j \in [1, m]$, let m be the initial message, with fixed part $\mathsf{m}_{\mathtt{FIX}}$, and let $\sigma = (\pi, \sigma_{\mathtt{FIX}}, \sigma_{\mathtt{FULL}}, \mathsf{ADM}, \widetilde{\mathbf{pk}}_{\text{SAN}}, \mathsf{w}_i, \mathbf{reg}_{sa})$ be a signature on m and MOD be instructions for a new message m'. First of all, if $\mathbf{pk}_{\mathsf{san}}[j] \notin \widetilde{\mathbf{pk}}_{\text{SAN}}$, then the algorithm returns $\perp$. Otherwise, she executes the following steps.

Proof of group membership. The sanitizer retrieves its value $\mathbf{reg}_{sa}[j, id_{\mathsf{m}}]$ in $\mathbf{reg}_{sa}$. She executes the GSKGn procedure on input $\mathbf{upk}_{sa}[j]$ and $\mathbf{reg}_{sa}[j, id_{\mathsf{m}}]$

to compute $\mathsf{\textbf{msk}}[j, id_{\mathsf{m}}]$. Finally, the sanitizer produces a new group signature $\sigma'_{\mathtt{full}} = \mathcal{GS}_2.\text{GSIGN}(\mathsf{\textbf{msk}}[j, id_{\mathsf{m}}], \mathsf{m}')$ on, behalf of the group of authorized entities to modified this message.

Proof of validity. As a sanitizer, she next produces a proof π' but as a sanitizer, which corresponds to the following:

- $\text{POK}(\mathsf{\textbf{msk}}[j, id_{\mathsf{m}}], \mathsf{\textbf{upk}}_{sa}[i], \mathsf{\textbf{reg}}_{sa}[j, id_{\mathsf{m}}])$:
  $\mathsf{\textbf{msk}}[j, id_{\mathsf{m}}] = \mathcal{GS}_2.\text{GSKGN}(\mathsf{\textbf{upk}}_{sa}[j],\ \mathsf{\textbf{reg}}_{sa}[j, id_{\mathsf{m}}]))$; and
- $\text{POK}(\mathsf{\textbf{msk}}[j, id_{\mathsf{m}}] : \sigma_{\mathtt{full}} = \mathcal{GS}_2.\text{GSIGN}(\mathsf{\textbf{msk}}[j, id_{\mathsf{m}}], \mathsf{m}'))$; and
- $\text{TPOK}(\mathsf{\textbf{upk}}_{sa}[j],\ \mathsf{\textbf{uc}}_{sa}[j] : \exists \mathsf{\textbf{sk}} \in \{\mathsf{cask}_{\mathsf{si}},\ \mathsf{cask}_{\mathsf{sa}}\} | \mathsf{\textbf{uc}}_{sa}[j] = \mathcal{S}.\text{SIGN}(\mathsf{\textbf{upk}}_{sa}[j],$ $\mathsf{\textbf{sk}}))$.

The resulting signature is finally $\sigma' = (\pi', \sigma_{\mathtt{FIX}}, \sigma'_{\mathtt{FULL}}, \mathsf{ADM}, \widetilde{\mathsf{\textbf{pk}}}_{\text{SAN}}, \mathsf{w}_i, \mathsf{\textbf{reg}}_{sa})$.

6.4 Verification and Opening Procedures

Verification. On input a signature $\sigma = (\pi, \sigma_{\mathtt{FIX}}, \sigma_{\mathtt{FULL}}, \mathsf{ADM},\ \widetilde{\mathsf{\textbf{pk}}}_{\text{SAN}}, \mathsf{w}_i,\ \mathsf{\textbf{reg}}_{sa})$ on a message m, the verification procedure simply checks both signatures $\sigma_{\mathtt{FIX}}$ and $\sigma_{\mathtt{FULL}}$ and the whole proof π. If all is correct, she outputs 1, otherwise, 0.

Opening. We finally describe the different opening procedures on input a signature σ as defined above. First of all, the AlgOpen procedure is simply executed by using the trap of the trapdoor or proof as shown in Section 5, and using the key $\mathsf{osk}_{\text{ALG}}$. The FindOri procedure is the execution of the $\mathcal{GS}_1.\text{OPEN}$ algorithm related to the group signature scheme for signers. Next, the SigOpen is executed as described in [9], by using the pseudo-random function and the opening algorithm of the group signature scheme. The FullOpen algorithm is simply the execution of $\mathcal{GS}_2.\text{OPEN}$.

6.5 Dealing with the Group Anonymity

The group anonymity states that the anonymity of sanitizers (resp. signers) is preserved, except for the sanitizers (resp. signers). In this case and regarding the above construction, each sanitizer (resp. signer) should be able to independently decrypt the same message: we need a *multi receiver encryption scheme.*

In such a scheme, a designated authority having a master key generates all the "receivers" (sanitizers or signers) secret keys. Such concept already exists, under the name of broadcast encryption [15] when it includes a revocation mechanism, traitor tracing [7] when it treats the case of fraudulous receivers, multi-recipient encryption [2] when several messages are encrypted for several recipients. Our need is close to public key traitor tracing scheme, except we do not necessarily need a tracing procedure. An example, based on the work from Boneh and Franklin [7], is given in Appendix B.

6.6 Security Theorem

We finally give the following security theorem, for which the proof can be found in Appendix D.

Theorem 1. *Our full transparent and fully anonymous multi-players sanitizable signature verifies all the required security properties, assuming that the used group signature, the pseudo-random function, the signature scheme (underlying the trapdoor or proof) and the trapdoor or proof are secure.*

References

1. Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In *ESORICS 2005*, volume 3679 of *Lecture Notes in Computer Science*, pages 159–177. Springer, 2005.
2. Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. Randomness re-use in multi-recipient encryption schemeas. In *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 85–99. Springer, 2003.
3. Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer, 2003.
4. Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *J. Cryptology*, 16(3):185–215, 2003.
5. Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: the case of dynamic groups. In *CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2005.
6. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.
7. Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In *CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 1999.
8. Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In *Public Key Cryptography 2009*, volume 5443 of *Lecture Notes in Computer Science*, pages 317–336. Springer, 2009.
9. Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 444–461. Springer, 2010.
10. Sébastien Canard and Amandine Jambert. On extended sanitizable signature schemes. In *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2010.
11. Sébastien Canard, Fabien Laguillaumie, and Michel Milhau. Trapdoor sanitizable signatures and their application to content protection. In *ACNS 2008*, volume 5037 of *Lecture Notes in Computer Science*, pages 258–276. Springer, 2008.
12. D. Chaum and T. Pedersen. Wallet Databases with Observers. In *Advances in Cryptology - Crypto '92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag.

13. Ronald Cramer, Ivan Damgrard, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.
14. Cécile Delerablée and David Pointcheval. Dynamic fully anonymous short group signatures. In *VIETCRYPT 2006*, volume 4341 of *Lecture Notes in Computer Science*, pages 193–210. Springer, 2006.
15. Amos Fiat and Moni Naor. Broadcast encryption. In *CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer, 1993.
16. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.
17. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
18. Marek Klonowski and Anna Lauks. Extended sanitizable signatures. In *ICISC 2006*, volume 4296 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2006.
19. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.
20. Alfredo De Santis, Giovanni Di Crescenzo, Giuseppe Persiano, and Moti Yung. On monotone formula closure of SZK. In *FOCS'94*, pages 454–465. IEEE, 1994.

A More on Multi-Players Sanitizable Signatures Schemes

A.1 Correctness

The correctness property for these algorithms states that all of them should be correct, from the verification to the different opening algorithms. This means that honestly generated signatures will always be accepted and honestly generated proofs and openings on correct signatures will always be accepted.

Signing Correctness. For any parameters λ, n, $m \in \mathbb{N}$, any keys $(\mathsf{gpk}, \mathsf{isk}, \mathsf{osk}) \leftarrow \textsc{Setup}(1^\lambda)$ and vectors $(\mathbf{sk}_{\text{SIG}}, \mathbf{pk}_{\text{SIG}}) \leftarrow \textsc{SigKG}(1^n, 1^\lambda, \mathsf{isk}_{\text{SIG}})$ and $(\mathbf{sk}_{\text{SAN}}, \mathbf{pk}_{\text{SAN}}) \leftarrow \textsc{SanKG}(1^m, 1^\lambda, \mathsf{isk}_{\text{SAN}})$, any message $\mathsf{m} \in \{0, 1\}^*$, any admissible instructions ADM, any signer $i \in [1..n]$, any subset of sanitizers $\widetilde{\mathbf{pk}}_{\text{SAN}} \subseteq \mathbf{pk}_{\text{SAN}}$ and any signature $\sigma \leftarrow \textsc{Sign}(\mathsf{m}, \mathbf{sk}_{\text{SIG}}[i], \widetilde{\mathbf{pk}}_{\text{SAN}}, \mathsf{ADM}, \mathsf{PK})$, we have $\textsc{Verify}(\mathsf{m}, \sigma, \mathsf{PK}) = 1$.

Sanitizing Correctness. For any parameters λ, n, $m \in \mathbb{N}$, any keys $(\mathsf{gpk}, \mathsf{isk}, \mathsf{osk}) \leftarrow \textsc{Setup}(1^\lambda)$ and vectors $(\mathbf{sk}_{\text{SIG}}, \mathbf{pk}_{\text{SIG}}) \leftarrow \textsc{SigKG}(1^n, 1^\lambda, \mathsf{isk}_{\text{SIG}})$ and $(\mathbf{sk}_{\text{SAN}}, \mathbf{pk}_{\text{SAN}}) \leftarrow \textsc{SanKG}(1^m, 1^\lambda, \mathsf{isk}_{\text{SAN}})$, any message $\mathsf{m} \in \{0, 1\}^*$, any σ such that $\textsc{Verify}(\mathsf{m}, \sigma, \mathsf{PK}) = 1$, any modification instructions MOD matching $\mathsf{ADM} \in \sigma$, any sanitizer $j \in [1..m]$ such that $\mathbf{pk}_{\text{SAN}}[j] \in \widetilde{\mathbf{pk}}_{\text{SAN}} \in \sigma$ and any pair $(\mathsf{m}', \sigma') \leftarrow \textsc{Sanitize}(\mathsf{m}, \sigma, \mathbf{sk}_{\text{SAN}}[j], \mathsf{MOD}, \mathsf{PK})$, we have $\textsc{Verify}(\mathsf{m}', \sigma', \mathsf{PK}) = 1$.

Origin Finding Correctness. For any parameters λ, n, $m \in \mathbb{N}$, any keys (gpk, isk, osk) $\leftarrow$ $\textsc{Setup}(1^\lambda)$ and vectors ($\mathbf{sk}_{\textsc{sig}}$, $\mathbf{pk}_{\textsc{sig}}$) $\leftarrow$ $\textsc{SigKG}(1^n, 1^\lambda, \mathsf{isk}_{\textsc{sig}})$ and ($\mathbf{sk}_{\textsc{san}}$, $\mathbf{pk}_{\textsc{san}}$) $\leftarrow$ $\textsc{SanKG}(1^m, 1^\lambda, \mathsf{isk}_{\textsc{san}})$, any message $\mathsf{m} \in \{0, 1\}^*$, any admissible instructions ADM, any signer $i \in [1..n]$, any subset of sanitizers $\widetilde{\mathbf{pk}}_{\textsc{san}} \subseteq \mathbf{pk}_{\textsc{san}}$ and any signature $\sigma \leftarrow \textsc{Sign}(\mathsf{m}, \mathbf{sk}_{\textsc{sig}}[i], \widetilde{\mathbf{pk}}_{\textsc{san}}, \mathsf{ADM}, \mathsf{PK})$, we have

1. For any triplet ($\textsc{ori}$, $I_{\textsc{ori}}$, $\tau_{\textsc{ori}}$) $\leftarrow$ $\textsc{FindOri}(\mathsf{m}, \sigma, \mathsf{osk}_{\textsc{ori}}, \mathsf{PK})$, we have $I_{\textsc{ori}} = (\mathsf{sig}, i_{\textsc{ori}})$, $i_{\textsc{ori}} = i$ and $\textsc{Judge}(\mathsf{m}, \sigma, (\textsc{ori}, I_{\textsc{ori}}, \tau_{\textsc{ori}}), \mathsf{PK}) = 1$.
2. For any modification instructions MOD matching ADM, any sanitizer $j \in [1..m]$ such that $\mathbf{pk}_{\textsc{san}}[j] \in \widetilde{\mathbf{pk}}_{\textsc{san}}$, any pair ($\mathsf{m}'$, σ') $\leftarrow$ $\textsc{Sanitize}(\mathsf{m}, \sigma, \mathbf{sk}_{\textsc{san}}[j], \mathsf{MOD}, \mathsf{PK})$ and any triplet ($\textsc{ori}$, $I_{\textsc{ori}}$, $\tau_{\textsc{ori}}$) $\leftarrow$ $\textsc{FindOri}(\mathsf{m}', \sigma', \mathsf{osk}_{\textsc{ori}}, \mathsf{PK})$, we have $I_{\textsc{ori}} = (\mathsf{sig}, i_{\textsc{ori}})$, $i_{\textsc{ori}} = i$ and $\textsc{Judge}(\mathsf{m}', \sigma', (\textsc{ori}, I_{\textsc{ori}}, \tau_{\textsc{ori}}), \mathsf{PK}) = 1$.

Algorithm Opening Correctness. For any parameters λ, n, $m \in \mathbb{N}$, any keys (gpk, isk, osk) $\leftarrow$ $\textsc{Setup}(1^\lambda)$ and vectors ($\mathbf{sk}_{\textsc{sig}}$, $\mathbf{pk}_{\textsc{sig}}$) $\leftarrow$ $\textsc{SigKG}(1^n, 1^\lambda, \mathsf{isk}_{\textsc{sig}})$ and ($\mathbf{sk}_{\textsc{san}}$, $\mathbf{pk}_{\textsc{san}}$) $\leftarrow$ $\textsc{SanKG}(1^m, 1^\lambda, \mathsf{isk}_{\textsc{san}})$, any message $\mathsf{m} \in \{0, 1\}^*$, any admissible instructions ADM, any signer $i \in [1..n]$, any subset of sanitizers $\widetilde{\mathbf{pk}}_{\textsc{san}} \subseteq \mathbf{pk}_{\textsc{san}}$ and any signature $\sigma \leftarrow \textsc{Sign}(\mathsf{m}, \mathbf{sk}_{\textsc{sig}}[i], \widetilde{\mathbf{pk}}_{\textsc{san}}, \mathsf{ADM}, \mathsf{PK})$, we have

1. For any triplet ($\textsc{alg}$, $I_{\textsc{alg}}$, $\tau_{\textsc{alg}}$) $\leftarrow$ $\textsc{AlgOpen}(\mathsf{m}, \sigma, \mathsf{osk}_{\textsc{alg}}, \mathsf{PK})$, we have $I_{\textsc{alg}} = \mathsf{sig}$ and $\textsc{Judge}(\mathsf{m}, \sigma, (\textsc{alg}, I_{\textsc{alg}}, \tau_{\textsc{alg}}), \mathsf{PK}) = 1$.
2. For any modification instructions MOD matching ADM, any sanitizer $j \in [1..m]$ such that $\mathbf{pk}_{\textsc{san}}[j] \in \widetilde{\mathbf{pk}}_{\textsc{san}}$, any pair ($\mathsf{m}'$, σ') $\leftarrow$ $\textsc{Sanitize}(\mathsf{m}, \sigma, \mathbf{sk}_{\textsc{san}}[j], \mathsf{MOD}, \mathsf{PK})$ and any triplet ($\textsc{alg}$, $I_{\textsc{alg}}$, $\tau_{\textsc{alg}}$) $\leftarrow$ $\textsc{AlgOpen}(\mathsf{m}', \sigma', \mathsf{osk}_{\textsc{alg}}, \mathsf{PK})$, we have $I_{\textsc{alg}} = \mathsf{san}$ and $\textsc{Judge}(\mathsf{m}', \sigma', (\textsc{alg}, I_{\textsc{alg}}, \tau_{\textsc{alg}}), \mathsf{PK}) = 1$.

Full Opening Correctness. For any parameters λ, n, $m \in \mathbb{N}$, any keys (gpk, isk, osk) $\leftarrow$ $\textsc{Setup}(1^\lambda)$ and vectors ($\mathbf{sk}_{\textsc{sig}}$, $\mathbf{pk}_{\textsc{sig}}$) $\leftarrow$ $\textsc{SigKG}(1^n, 1^\lambda, \mathsf{isk}_{\textsc{sig}})$ and ($\mathbf{sk}_{\textsc{san}}$, $\mathbf{pk}_{\textsc{san}}$) $\leftarrow$ $\textsc{SanKG}(1^m, 1^\lambda, \mathsf{isk}_{\textsc{san}})$, any message $\mathsf{m} \in \{0, 1\}^*$, any admissible instructions ADM, any signer $i \in [1..n]$, any subset of sanitizers $\widetilde{\mathbf{pk}}_{\textsc{san}} \subseteq \mathbf{pk}_{\textsc{san}}$ and any signature $\sigma \leftarrow \textsc{Sign}(\mathsf{m}, \mathbf{sk}_{\textsc{sig}}[i], \widetilde{\mathbf{pk}}_{\textsc{san}}, \mathsf{ADM}, \mathsf{PK})$, we have

1. For any triplet ($\textsc{full}$, $I_{\textsc{full}}$, $\tau_{\textsc{full}}$) $\leftarrow$ $\textsc{FullOpen}(\mathsf{m}, \sigma, \mathsf{osk}_{\textsc{full}}, \mathsf{PK})$, we have $I_{\textsc{full}} = (\mathsf{sig}, i_{\textsc{full}})$, $i_{\textsc{full}} = i$ and $\textsc{Judge}(\mathsf{m}, \sigma, (\textsc{full}, I_{\textsc{full}}, \tau_{\textsc{full}}), \mathsf{PK}) = 1$.
2. For any modification instructions MOD matching ADM, any sanitizer $j \in [1..m]$ such that $\mathbf{pk}_{\textsc{san}}[j] \in \widetilde{\mathbf{pk}}_{\textsc{san}}$, any pair ($\mathsf{m}'$, σ') $\leftarrow$ $\textsc{Sanitize}(\mathsf{m}, \sigma, \mathbf{sk}_{\textsc{san}}[j], \mathsf{MOD}, \mathsf{PK})$ and any triplet ($\textsc{full}$, $I_{\textsc{full}}$, $\tau_{\textsc{full}}$) $\leftarrow$ $\textsc{FullOpen}(\mathsf{m}', \sigma', \mathsf{osk}_{\textsc{full}}, \mathsf{PK})$, we have $I_{\textsc{full}} = (\mathsf{san}, j_{\textsc{full}})$, $j_{\textsc{full}} = j$ and $\textsc{Judge}(\mathsf{m}', \sigma', (\textsc{full}, I_{\textsc{full}}, \tau_{\textsc{full}}), \mathsf{PK}) = 1$.

Signature Opening Correctness. For any parameters λ, n, $m \in \mathbb{N}$, any keys (gpk, isk, osk) $\leftarrow$ $\textsc{Setup}(1^\lambda)$ and vectors ($\mathbf{sk}_{\textsc{sig}}$, $\mathbf{pk}_{\textsc{sig}}$) $\leftarrow$ $\textsc{SigKG}(1^n, 1^\lambda, \mathsf{isk}_{\textsc{sig}})$ and ($\mathbf{sk}_{\textsc{san}}$, $\mathbf{pk}_{\textsc{san}}$) $\leftarrow$ $\textsc{SanKG}(1^m, 1^\lambda, \mathsf{isk}_{\textsc{san}})$, any message $\mathsf{m} \in \{0, 1\}^*$, any admissible instructions ADM, any signers $i, \tilde{i} \in [1..n]$, any subset of sanitizers

$\widetilde{\mathbf{pk}}_{\text{SAN}} \subseteq \mathbf{pk}_{\text{SAN}}$ and any signature $\sigma \leftarrow \textsc{Sign}(\mathsf{m}, \mathbf{sk}_{\text{SIG}}[i], \widetilde{\mathbf{pk}}_{\text{SAN}}, \mathsf{ADM}, \mathsf{PK})$, any (polynomially many) $\mathsf{DB} := \{(\mathsf{m}_k, \sigma_k)\}_{k=1}^{q}$ such that $\textsc{Verify}(\mathsf{m}_k, \sigma_k, \mathsf{PK}) = 1$ for all $k \in [1..q]$ and $(\mathsf{m}, \sigma) \in \mathsf{DB}$, any $(\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}})$ such that $\textsc{Judge}(\mathsf{m}, \sigma, (\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}), \mathsf{PK}) = 1$, we have

1. For any triplet $(\text{SIG}, I_{\text{SIG}}, \tau_{\text{SIG}}) \leftarrow \textsc{SigOpen}(\mathsf{m}, \sigma, (\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}), \mathbf{sk}_{\text{SIG}}[\tilde{i}], \mathsf{PK}, \mathsf{DB})$, we have $I_{\text{SIG}} = I_{\text{ORI}}$ and $\textsc{Judge}(\mathsf{m}, \sigma, (\text{SIG}, I_{\text{SIG}}, \tau_{\text{SIG}}), \mathsf{PK}) = 1$.
2. For any modification instructions MOD matching ADM, any sanitizer $j \in [1..m]$ such that $\mathbf{pk}_{\text{SAN}}[j] \in \widetilde{\mathbf{pk}}_{\text{SAN}}$ and any pair $(\mathsf{m}', \sigma') \leftarrow \textsc{Sanitize}(\mathsf{m}, \sigma, \mathbf{sk}_{\text{SAN}}[j], \mathsf{MOD}, \mathsf{PK})$, any triplet $(\text{SIG}, I_{\text{SIG}}, \tau_{\text{SIG}}) \leftarrow \textsc{SigOpen}(\mathsf{m}', \sigma', (\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}), \mathbf{sk}_{\text{SIG}}[\tilde{i}], \mathsf{PK}, \mathsf{DB})$, we have $I_{\text{SIG}} = (\text{SAN}, 0)$ and $\textsc{Judge}(\mathsf{m}', \sigma', (\text{SIG}, I_{\text{SIG}}, \tau_{\text{SIG}}), \mathsf{PK}) = 1$.

Remark 1. For sake of simplicity, we assume that any sanitization procedure changes at least one block of messages. We leave a void when no modification is done. In some schemes or concrete cases, a sanitization that does nothing might trace preferably to the sanitizer or to the signer.

B Multi Receiver Encryption Schemes

We have seen that in some cases, several actors should be able to independently decrypt the same message, which leads to the concept of multi receiver encryption scheme. We here give an example based on the work from [7]. Let λ be a security parameter. $\textsc{KeyGen}$: let $\mathbb{G}$ be a cyclic group of prime order q and let g be a generator of $\mathbb{G}$. The trusted authority chooses at random $w_1, \ldots, w_\lambda \in \mathbb{Z}_p^*$ and define $h_j = g^{w_j}$ for all $j \in [1, \lambda]$. It next chooses at random $\alpha \in \mathbb{Z}_p^*$. After that, for each $i \in [1, n]$, it computes $\alpha_1^{(i)}, \ldots, \alpha_\lambda^{(i)} \in \mathbb{Z}_p^*$ such that $\alpha = \sum_{j=1}^{\lambda} w_j \alpha_j^{(i)} \pmod p$ and such that the $\{\alpha_j^{(i)}\}_{j=1}^{\lambda}$ are all different. It finally computes $a = g^\alpha = \prod_{j=1}^{\lambda} h_j^{\alpha_j^{(i)}}$. The user i finally gets her secret key $\{\alpha_j^{(i)}\}_{j=1}^{\lambda}$ and the global public key is $(p, g, h_1, \ldots, h_\lambda, a)$. $\textsc{Enc}$: the encryption algorithm takes as input the public key and a message $m \in \mathbb{G}$. It chooses at random a $r \in \mathbb{Z}_p^*$ and computes $T = ma^r$ and $(T_1 = h_1^r, \ldots, T_\lambda = h_\lambda^r)$. $\textsc{Dec}$: the decryption algorithm takes as input one secret key $\{\alpha_j^{(i)}\}_{j=1}^{\lambda}$ and a ciphertext $(T, \{T_j\}_{j=1}^{\lambda})$. m is retrieved as $m = T / \prod_{j=1}^{\lambda} (T_j)^{\alpha_j^{i)}}$.

C Concrete Construction for Trapdoor Or Proofs

Let $\mathbb{G}$ be a group of prime order p. Let u, h be random generators of $\mathbb{G}$ and let v and z be two elements of $\mathbb{G}$. We want to design the trapdoor or proof denoted $\textsc{PoK}(\alpha : v = u^\alpha \overset{(t)}{\vee} z = h^\alpha)$.

Our solution makes use of a homomorphic encryption scheme $\pi = (\textsc{KeyGen}, \textsc{Enc}, \textsc{Dec})$ such that the trapdoor of our construction is the decryption key dk. The encryption public key is $\mathsf{ek} = a$ and the corresponding secret key is $\alpha \in \mathbb{Z}_p^*$

such that $a = d^{\alpha}$ where $d \in \mathbb{G}$. A prover having access to *e.g.* the discrete logarithm $x \in \mathbb{Z}_p^*$ of v in base u, that is $v = u^x$, can produce a *trapdoor or proof* as follows, with the ElGamal encryption scheme as an concrete instantiation (see [16]).

1. Encrypt v and u as $c_v = \textsc{Enc}(\mathsf{ek}, v)$ and $c_u = \textsc{Enc}(\mathsf{ek}, u)$. In the ElGamal case, we obtain $c_v = (t_1 = va^w, t_2 = d^w)$ and $c_u = (t_3 = ua^r, t_4 = d^r)$ where $w, r \in \mathbb{Z}_p^*$.
2. Produce a (traditional) proof of knowledge on x, r and w such that:
   (a) the pair of encrypted values corresponds to either (v, u) or (z, h). To this aim, we produce the set membership proof that the couple $(\textsc{Dec}(\mathsf{dk}, c_v)$, $\textsc{Dec}(\mathsf{dk},\ c_u)) \in \{(v, u), (z, h)\}$. Considering the ElGamal encryption scheme, this consists in producing the following standard ZKPK:
   $$\textsc{Pok}\Big(w, r : \big((t_1/v = a^w \wedge t_3/u = a^r) \vee$$
   $$(t_1/z = a^w \wedge t_3/h = a^r)\big) \wedge t_2 = d^w \wedge t_4 = d^r\Big);$$
   (b) using the encrypted value (which satisfies the relation $v = u^x$) and the homomorphic property of the encryption scheme, it is done by producing the proof of knowledge of x such that $\textsc{Enc}(\mathsf{ek}, v) = \textsc{Enc}(\mathsf{ek}, u)^x$ (that is $c_v = c_u^x$). Considering the ElGamal scheme, we remark that $t_3^x = u^x a^{rx} = va^{rx} = t_1 a^{rx-w}$. We thus need to prove that $t_3^x = t_1 a^{rx} a^{-w}$ and that $\bar{r} = rx$ is the multiplication of r and x, which is done by using t_4. Next, as it is redundant to prove twice the knowledge of r such that $t_4 = d^r$, the final *trapdoor or proof* is composed of (t_1, t_2, t_3, t_4) and the following proof of knowledge:
   $$V = \textsc{Pok}\big(w, r, x, \bar{r} : \big((\tfrac{t_1}{v} = a^w \wedge \tfrac{t_3}{u} = a^r) \vee (\tfrac{t_1}{z} = a^w \wedge \tfrac{t_3}{h} = a^r)\big)$$
   $$\wedge t_2 = d^w \wedge t_4 = d^r \wedge t_1 = t_3^x a^{-\bar{r}} a^w \wedge 1 = t_4^x d^{-\bar{r}}\Big)$$

Anyone in possession of α can retrieve the encrypted pair (v, u) and obtain the known discrete logarithm. We here present the more general case where we need to encrypt both u and v. When this is relevant, the encryption of u or v (but not both) can be replaced by a commitment. We have given the solution for an or proof for a discrete logarithm. As in [13, 20], the (trapdoor) or proof for a representation can also be treated similarly. We do not detailed the case of a representation but we will use it in the following section.

D Security Proof for Our Main Scheme

We here prove that our main construction described in Section 6 is a secure fully transparent and fully anonymous multi-players sanitizable signature scheme. Note that, below, we do not treat similarly both group signatures. On one side, as we have not modified the group signature $\sigma_{\mathtt{FIX}}$, we reduce our property to the security of the group signature. On the other side, the signature of knowledge underlying the group signature $\sigma_{\mathtt{FULL}}$ is modified to embed our trapdoor or proof. As a consequence, we do not reduce the security of our construction to the security of $\sigma_{\mathtt{FULL}}$ but of the underlying building blocks.

Immutability. An adversary against the immutability property outputs a message-signature pair and a sanitizer's public key $\mathbf{pk}_{\text{SAN}}[j^*]$ accepted if either one non-admissible part of the message has been sanitized or $\mathbf{pk}_{\text{SAN}}[j^*]$ is not among the public keys of the sanitizers that are allowed to sanitize this pair. We show it contradicts either the traceability or the non-frameability of our group signature for the group of signers (and thus the unforgeability of the group signature scheme, which is coherent with the result given in [9]). We define two different games we repeatedly play at random with an adversary $\mathcal{A}$ until one succeeds.

Traceability. We first interact with $\mathcal{A}$ to construct a machine $\mathcal{B}_t$ against the traceability of the group signature scheme. $\mathcal{B}_t$ receives the global parameters and the opener's secret key $(\zeta, \tilde{\zeta})$ (which will become the key for $\mathcal{O}_{\text{ORI}}$). It creates the keys of $\mathcal{O}_{\text{FULL}}$ and $\mathcal{O}_{\text{ALG}}$ and generates the group of n signers by interacting with its challenger to obtain their public group member keys, using the oracle at its disposal (denoted AddU in [5]). Each signer finally obtains a decryption key related to $\mathcal{O}_{\text{ALG}}$ and the multi receiver encryption scheme (see Section 6.1) and the certification, by $\mathcal{CA}$, of her key v_i, using the secret key of both $\mathcal{O}_{\text{ALG}}$ and $\mathcal{CA}$. $\mathcal{B}_t$ gives the global public key of the sanitizable signature scheme to $\mathcal{A}$, the key osk and the public key of all signers. Next (i) if $\mathcal{A}_{\mathsf{ch}}$ wants to corrupt a signer, $\mathcal{B}_t$ uses its own corruption oracle to obtain the group member secret keys of the requested user. It next adds other keys related to this him (multi receiver decryption key and certified ones) and gives keys to adversary. (ii) If $\mathcal{A}_{\mathsf{ch}}$ asks for a SIGN oracle with a subset of sanitizers, $\mathcal{B}_t$ first executes the step 2 of the SIGN procedure given in Section 6.2 to create the group member keys of all the sanitizers given by $\mathcal{A}_{\mathsf{ch}}$ plus the chosen signer. $\mathcal{B}_t$ next asks its own challenger for a group signature on behalf of the chosen signer, which gives it the value U. It finally executes the resulting steps of the signing algorithm, as described in Section 6.2, as it possesses all the needed keys for that. (iii) A request to the SIGOPEN oracle is carried out as described in Section 6.4. (iv) Other oracles can easily be simulated. $\mathcal{A}_{\mathsf{ch}}$ eventually outputs a sanitizer j^*, a message m^* and a valid signature $\sigma^* = (\sigma^*_{\mathtt{FIX}}, \sigma^*_{\mathtt{FULL}}, \mathsf{ADM}^*,\ J^*, \mathsf{w}^*_i, \mathbf{key}^*)$. Let $\mathsf{m}^*_{\mathtt{FIX}}$ be the fixed part of the message m^* and let $(\text{ORI}, (\mathsf{sig}, I_{\text{ORI}}), \tau_{\text{ORI}}) = \text{FINDORI}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{ORI}})$. If $\mathcal{A}_{\mathsf{ch}}$ succeeds, then $\sigma^*_{\mathtt{FIX}}$ is a valid group signature on the message $\mathsf{m}^*_{\text{SIG}} = (\mathsf{m}^*_{\mathtt{FIX}}, \mathsf{ADM}^*,\ \mathbf{pk}^{(J^*)}_{\text{SAN}}, \mathbf{key}^*)$. As the initial underlying signer is not corrupted, either $(\mathsf{m}^*_{\text{SIG}}, \sigma^*_{\mathtt{FIX}})$ comes from the group signing oracle and has been requested by $\mathcal{B}_t$, or this is a forgery. In the first case, let $(\mathsf{m}_{\mathtt{SIG,k}}, \sigma_{FIX,k})$ be the related oracle call, with $\mathsf{m}_{\mathtt{SIG,k}} = (\mathsf{m}_{FIX,k}, \mathsf{ADM}_k, \mathbf{pk}^{(J_k)}_{\text{SAN}},\ \mathbf{key}_k)$. We have $\mathsf{ADM}^* = \mathsf{ADM}_k$ and $\mathbf{pk}^{(J^*)}_{\text{SAN}} = \mathbf{pk}^{(J_k)}_{\text{SAN}}$, which contradicts the fact that $\mathcal{A}$ is successful since the conditions state that $\exists l_0 \in [1, t] : \mathsf{m}_{l_0} \in \mathsf{MOD}, l_0 \notin \mathsf{ADM}$ or $\forall k \in [1, q]$, if $\mathsf{ADM}(\mathsf{m}^*, \mathsf{m}_k) = 1$, then $\mathbf{pk}_{\text{SAN}}[j^*] \notin \widetilde{\mathbf{pk}}_{\text{SAN},k}$. Thus $(\mathsf{m}^*_{\text{SIG}}, \sigma^*_{\mathtt{FIX}})$ is a forgery and $\mathcal{B}_t$ can send it back to its own challenger. Three cases may happen, depending on the output $(\text{ORI}, (\mathsf{sig}, I_{\text{ORI}}), \tau_{\text{ORI}})$ of the FINDORI procedure. (I) $I_{\text{ORI}} = 0$: the output of $\mathcal{B}_t$ to its challenger is successful. (II) $I_{\text{ORI}} \neq 0$ and $\text{JUDGE}(\mathsf{m}^*, \sigma^*, (\text{ORI}, (\mathsf{sig}, I_{\text{ORI}}), \tau_{\text{ORI}}), \mathsf{PK}) = 0$: the output of $\mathcal{B}_t$ to its challenger is successful. (III) $I_{\text{ORI}} \neq 0$ and $\text{JUDGE}(\mathsf{m}^*, \sigma^*, (\text{ORI}, (\mathsf{sig}, I_{\text{ORI}}), \tau_{\text{ORI}}), \mathsf{PK}) = 1$: we cannot conclude and output $\perp$.

Non-frameability. We next interact with $\mathcal{A}$ to construct a machine $\mathcal{B}_n$ against the non-frameability of the group signature scheme. $\mathcal{B}_n$ receives the parameters and keys, and creates keys of $\mathcal{O}_{\text{FULL}}$ and $\mathcal{O}_{\text{ALG}}$. Next $\mathcal{B}_n$ generates the group of n signers by first interacting with its challenger to obtain their public group member keys, playing the role of the group manager and using the oracle at its disposal (denoted SndToU in [5] which allows an adversary to send messages to users, in the name of a corrupted issuer). Similarly to the above experiment for the traceability property, each signer finally obtains a decryption key and the certification of her key v_i. The opener's secret osk is created. $\mathcal{B}_n$ finally gives to $\mathcal{A}$ the global public key, the key osk and the public key of all signers. $\mathcal{A}_{\mathsf{ch}}$ next interacts with $\mathcal{B}_n$ similarly to the above traceability game. $\mathcal{A}_{\mathsf{ch}}$ eventually outputs a sanitizer j^*, a message m^* and a valid signature $\sigma^* = (\sigma^*_{\mathtt{FIX}}, \sigma^*_{\mathtt{FULL}}, \mathsf{ADM}^*, J^*, \mathsf{w}^*_i, \mathbf{key}^*)$. Let $\mathsf{m}^*_{\mathtt{FIX}}$ be the fixed part of the message m^* and let $(\text{ORI}, (\mathsf{sig}, I_{\text{ORI}}), \tau_{\text{ORI}}) = \text{FINDORI}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{SIG}})$. If $\mathcal{A}_{\mathsf{ch}}$ succeeds, then $\sigma^*_{\mathtt{FIX}}$ is a valid group signature on the message $\mathsf{m}^*_{\text{SIG}} = (\mathsf{m}^*_{\mathtt{FIX}}, \mathsf{ADM}^*, \mathbf{pk}^{(J^*)}_{\text{SAN}}, \mathbf{key}^*)$. Again, the output $(\mathsf{m}^*_{\text{SIG}}, \sigma^*_{\mathtt{FIX}})$ is a forgery (see above) and $\mathcal{B}_n$ can send it back to its own challenger. (I) $I_{\text{ORI}} = 0$: we cannot conclude and output $\perp$. (II) $I_{\text{ORI}} \neq 0$ and $\text{JUDGE}(\mathsf{m}^*, \sigma^*, \mathsf{gpk}, (\text{ORI}, (\mathsf{sig}, I_{\text{ORI}}), \tau_{\text{ORI}}), \mathsf{PK}) = 0$: we cannot conclude output $\perp$. (III) $I_{\text{ORI}} \neq 0$ and $\text{JUDGE}(\mathsf{m}^*, \sigma^*, \mathsf{gpk}, (\text{ORI}, (\mathsf{sig}, I_{\text{ORI}}), \tau_{\text{ORI}}), \mathsf{PK}) = 1$: the output of $\mathcal{B}_n$ to its challenger is successful.
Conclusion. The above simulation is perfect. Hence, the final output of the adversary against the immutability of our scheme is distributed identically to the real game and $\mathbf{Succ}^{\mathsf{imm}}_{\Pi,\mathcal{A}}(\lambda, n, m) \leqslant \mathbf{Succ}^{\mathsf{trac}}_{\mathcal{GS},\mathcal{A}}(\lambda) + \mathbf{Succ}^{\mathsf{nf}}_{\mathcal{GS},\mathcal{A}}(\lambda)$.

Traceability. An adversary against the traceability property outputs a message-signature pair, which is accepted if one of the two following conditions is verified: (i) $\text{FULLOPEN}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{FULL}}, \mathsf{PK}) = (\text{FULL}, I_{\text{FULL}}, \tau_{\text{FULL}})$ and either $\text{JUDGE}(\mathsf{m}^*, \sigma^*, (\text{FULL}, I_{\text{FULL}}, \tau_{\text{FULL}}, \mathsf{PK})) = 0$ or $I_{\text{FULL}} = 0$; (ii) $\text{FINDORI}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{ORI}}, \mathsf{PK}) = (\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}})$ and either $\text{JUDGE}(\mathsf{m}^*, \sigma^*, (\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}), \mathsf{PK}) = 0$ or $I_{\text{ORI}} = 0$. We show it contradicts either the traceability of the group signature (for the group of signers) or the unforgeability of the $\mathcal{CA}$'s signature scheme with extensions. We define two different games we repeatedly play at random with an adversary $\mathcal{A}$ until one succeeds.
Traceability. We first interact with $\mathcal{A}$ to construct a machine $\mathcal{B}_t$ against the traceability of the group signature scheme. Again, this part of the game is very close to the one given above for the immutability property. $\mathcal{B}_t$ receives the public key of the group signature scheme and the opener's secret key $(\zeta, \tilde{\zeta})$ (the secret key for $\mathcal{O}_{\text{ORI}}$) and next creates, according to Section 6.1, all the remaining keys, especially for $\mathcal{O}_{\text{FULL}}$ and $\mathcal{O}_{\text{ALG}}$. Next $\mathcal{B}_t$ interacts with $\mathcal{A}$ to enrol each signer in the group of signers. For this purpose, $\mathcal{B}_t$ uses the SndToI oracle (see [5]) and simply forwards the messages received from and for the adversary to this oracle. This way, the adversary $\mathcal{A}$ knows all the group member's secret keys (while this is not necessarily the case for our machine) and each of them is a member of the group. A request to the FINDORI (resp. FULLOPEN and ALGOPEN) oracle is carried out with $\mathsf{osk}_{\text{ORI}}$ (resp. $\mathsf{osk}_{\text{FULL}}$ and $\mathsf{osk}_{\text{ALG}}$) and using the procedure

given in Section 6.4. $\mathcal{A}_{\mathsf{ch}}$ eventually outputs a message m^* and a valid signature $\sigma^* = (\sigma^*_{\mathtt{FIX}}, \sigma^*_{\mathtt{FULL}}, \mathsf{ADM}^*, J^*, \mathsf{w}^*_i, \mathbf{key}^*)$ on m^*. We use the $\mathcal{O}_{\text{FULL}}$ and $\mathcal{O}_{\text{ORI}}$ keys to get $(\text{FULL}, (\mathsf{sig}, I_{\text{FULL}}), \tau_{\text{FULL}}) = \text{FULLOPEN}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{FULL}}, \mathsf{PK})$ and $(\text{ORI}, (\mathsf{sig}, I_{\text{ORI}}), \tau_{\text{ORI}}) = \text{FINDORI}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{ORI}}, \mathsf{PK})$. Two cases may happen. (I) $\text{JUDGE}(\mathsf{m}^*, \sigma^*, (\text{FULL}, I_{\text{FULL}}, \tau_{\text{FULL}}), \mathsf{PK}) = 0$ or $I_{\text{FULL}} = 0$. We cannot conclude and output $\perp$. (II) $\text{JUDGE}(\mathsf{m}^*, \sigma^*, (\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}), \mathsf{PK}) = 0$ or $I_{\text{ORI}} = 0$. $\mathcal{B}_t$ sends (m^*, σ^*) to its challenger, which is a successful output for the traceability game with non-negligible probability. As $\mathcal{A}$ wins its game with non-negligible probability, we must fall into one of the two cases.
Unforgeability. We next interact with $\mathcal{A}$ to construct a machine $\mathcal{B}_u$ against the unforgeability of the $\mathcal{CA}$'s signature scheme with extensions, denoted $\mathcal{S}$. $\mathcal{B}_u$ receives from its challenger the parameters for the signature scheme and the challenged public key w, which now becomes the public key of the certification authority $\mathcal{CA}$. $\mathcal{B}_u$ executes the different key generation phases with the sole constraint that it should use the above parameters. $\mathcal{A}$ may create keys in the corruption phase and interacts with $\mathcal{B}_u$ which uses the previously generated keys, except for the certification of the b_i (for the signers) and the b_j's (for the sanitizers) with the $\mathcal{CA}$'s key, for which $\mathcal{B}_u$ uses the signing oracle. A request to the FINDORI (resp. FULLOPEN and ALGOPEN) oracle is executed by using the secret key of $\mathcal{O}_{\text{ORI}}$ (resp. $\mathcal{O}_{\text{FULL}}$ and $\mathcal{O}_{\text{ALG}}$) and using the procedure given in Section 6.4. $\mathcal{A}_{\mathsf{ch}}$ eventually outputs a message m^* and a valid signature $\sigma^* = (\sigma^*_{\mathtt{FIX}}, \sigma^*_{\mathtt{FULL}}, \mathsf{ADM}^*, J^*, \mathsf{w}^*_i, \mathbf{key}^*)$ on m^*. We next use the keys of $\mathcal{O}_{\text{FULL}}$ and $\mathcal{O}_{\text{ORI}}$ to obtain both $(\text{FULL}, (\mathsf{sig}, I_{\text{FULL}}), \tau_{\text{FULL}}) = \text{FULLOPEN}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{FULL}}, \mathsf{PK})$ and $(\text{ORI}, (\mathsf{sig}, I_{\text{ORI}}), \tau_{\text{ORI}}) = \text{FINDORI}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\text{ORI}}, \mathsf{PK})$. Two cases may happen: either we have $\text{JUDGE}(\mathsf{m}^*, \sigma^*, (\text{FULL}, I_{\text{FULL}}, \tau_{\text{FULL}}), \mathsf{PK}) = 0$ and the or $I_{\text{FULL}} = 0$. Using standard techniques in the random oracle model, $\mathcal{B}_u$ extracts from the proof U'^* in $\sigma^*_{\mathtt{FULL}}$ the value B_0 (see the example of group signature schemes in *e.g.* [6, 14] ; those techniques generically apply to signatures produced by the Fiat-Shamir heuristic and extractability of witnesses is based on the forking lemma, cf. [14, 19] for more details). As U'^* is correct, B_0 is related to the same actor in I_{FULL}. As the adversary $\mathcal{A}$ wins the traceability game with non negligible probability, the value B_0 does not come from the signing oracle. Thus, $\mathcal{B}_u$ sends B_0 to its challenger, which is successful forgery with non-negligible probability. (II) $\text{JUDGE}(\mathsf{m}^*, \sigma^*, (\text{ORI}, I_{\text{ORI}}, \tau_{\text{ORI}}), \mathsf{PK}) = 0$ or $I_{\text{ORI}} = 0$. We cannot conclude and output $\perp$. Since $\mathcal{A}$ wins its game, we must fall into one of the two above conditions.
Conclusion. The above simulation is perfect. Hence, the final output of the adversary against the immutability of our scheme is distributed identically to the real game and $\mathbf{Succ}^{\mathsf{open}}_{\Pi,\mathcal{A}}(\lambda, n, m) \leqslant \mathbf{Succ}^{\mathsf{trac}}_{\mathcal{GS},\mathcal{A}}(\lambda) + \mathbf{Succ}^{\mathsf{ex\text{-}unf}}_{\mathcal{S},\mathcal{A}}(\lambda)$.

Sanitizer accountability. An adversary against the sanitizer's accountability outputs a message-signature pair, a signer's identity and, optionally, a proof $(I^*_{\text{ALG}}, \tau^*_{\text{ALG}})$ related to the ALGOPEN procedure. This output is accepted if the message-signature pair is valid, if neither the given signer nor the original signer are corrupted and if one of the two following conditions is verified: (i) $\text{JUDGE}(\mathsf{m}^*,$

σ^*, $(\textsc{alg}, I^*_{\textsc{alg}}, \tau^*_{\textsc{alg}})$, $\mathsf{PK}) = 0$ or $I_{\textsc{full}} = \mathsf{sig}$; (ii) $\textsc{Judge}(\mathsf{m}^*, \sigma^*, (\textsc{sig}, I_{\textsc{sig}}, \tau_{\textsc{sig}}), \mathsf{PK}) = 0$ or $I_{\textsc{sig}} = (\mathsf{sig}, \star)$ with $(\textsc{ori}, I_{\textsc{ori}}, \tau_{\textsc{ori}}) = \textsc{FindOri}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\textsc{ori}}, \mathsf{PK})$ and $(\textsc{sig}, I_{\textsc{sig}}, \tau_{\textsc{sig}}) = \textsc{SigOpen}(\mathsf{m}^*, \sigma^*, (\textsc{ori}, I_{\textsc{ori}}, \tau_{\textsc{ori}}), \mathbf{sk}_{\textsc{sig}}[i^*], \mathsf{PK}, \cdot)$. We show it contradicts either the traceability, the non-frameability of the used group signature or the double unforgeability of the $\mathcal{CA}$'s signature scheme. We define four different games we repeatedly play at random with an adversary $\mathcal{A}$ until one succeeds.

Traceability and Non-frameability. We first interact with $\mathcal{A}$ to construct a machine $\mathcal{B}_t$ (resp. $\mathcal{B}_n$) against the traceability (resp. non-frameability) of the group signature scheme. As this part of the proof is very close to the games described for the immutability property, we only give the most important steps and the conclusions. The machine $\mathcal{B}_t$ (resp. $\mathcal{B}_n$) runs the same key generation phase as the one described for the immutability related to the traceability (resp. non-frameability) property, as the oracles and keys are similar. At the end of this step, $\mathcal{B}_t$ (resp. $\mathcal{B}_n$) gives to $\mathcal{A}$ the global public key of the sanitizable signature scheme, the key osk and the public key of all signers. The next step consists in simulating the oracle for the adversary, which is again done as described for the immutability property. At the end, $\mathcal{A}_{\mathsf{ch}}$ outputs a message m^* and a valid signature $\sigma^* = (\sigma^*_{\mathtt{FIX}}, \sigma^*_{\mathtt{FULL}}, \mathsf{ADM}^*, J^*, \mathsf{w}^*_i, \mathbf{key}^*)$, a honest signer i^* and, optionally, a couple $(I^*_{\textsc{alg}}, \tau^*_{\textsc{alg}})$ output by the $\textsc{AlgOpen}$ procedure. Let us now focus on the signature σ^*, and more precisely on $\sigma^*_{\mathtt{FIX}}$. The condition of acceptance states that (m^*, σ^*) does not come from an oracle. (I) $(\mathsf{m}^*_{\textsc{sig}}, \sigma^*_{\mathtt{FIX}})$, with $\mathsf{m}^*_{\textsc{sig}} = (\mathsf{m}^*_{\mathtt{FIX}}, \mathsf{ADM}^*, \mathbf{pk}^{(J^*)}_{\textsc{san}}, \mathbf{key}^*)$, comes from an oracle (related to the $\textsc{Sign}$ one): we cannot conclude and output $\perp$. (II) $(\mathsf{m}^*_{\textsc{sig}}, \sigma^*_{\mathtt{FIX}})$ is a new couple which has not been output by an oracle. Let $(\textsc{ori}, (\mathsf{sig}, I_{\textsc{ori}}), \tau_{\textsc{ori}}) = \textsc{FindOri}(\mathsf{m}^*, \sigma^*, \mathsf{osk}_{\textsc{ori}}, \mathsf{PK})$. We have next three sub-cases. (II.i) $I_{\textsc{ori}} = 0$. $\mathcal{B}_t$ can use (m^*, σ^*) to break the traceability of the group signature scheme with non-negligible probability. (II.ii) $I_{\textsc{ori}} \neq 0$ and $\textsc{Judge}(\mathsf{m}^*, \sigma^*, (\textsc{ori}, (\mathsf{sig}, I_{\textsc{ori}}), \tau_{\textsc{ori}})) = 0$. $\mathcal{B}_t$ can use (m^*, σ^*) to break the traceability of the group signature scheme with non-negligible probability. (II.iii) $I_{\textsc{ori}} \neq 0$ and $\textsc{Judge}(\mathsf{m}^*, \sigma^*, (\textsc{ori}, (\mathsf{sig}, I_{\textsc{ori}}), \tau_{\textsc{ori}}), \mathsf{PK}) = 1$. $\mathcal{B}_n$ can use (m^*, σ^*) to break the non-frameability of the group signature scheme.

Unforgeability. We next interact with $\mathcal{A}$ to construct a machine $\mathcal{B}_u$ against the double unforgeability of the $\mathcal{CA}$'s signature scheme with extensions denoted $\mathcal{S}$. $\mathcal{B}_u$ first receives from its challenger the public parameters for the signature scheme, the challenged public key w and the signature scheme (0 or 1) which $\mathcal{B}_u$ has to break. The secret key related to w is now the public key of the certification authority $\mathcal{CA}$ and the challenge signature scheme now becomes the signature scheme for the certification of signer's keys (the other being the one for sanitizers). $\mathcal{B}_u$ executes the different key generation phases with the sole constraint that it should use the above parameters. The adversary $\mathcal{A}_{\mathsf{ch}}$ next interacts with the system which is simulated by $\mathcal{B}_u$ as previously described in Appendix D. $\mathcal{A}_{\mathsf{ch}}$ eventually outputs a message m^* and a valid signature $\sigma^* = (\sigma^*_{\mathtt{FIX}}, \sigma^*_{\mathtt{FULL}}, \mathsf{ADM}^*, J^*, \mathsf{w}^*_i, \mathbf{key}^*)$, a honest signer i^* and, optionally, a couple $(I^*_{\textsc{alg}}, \tau^*_{\textsc{alg}})$ output by the $\textsc{AlgOpen}$ procedure. Two cases may happen. (I)

$(\mathsf{m}^*_{\text{SIG}}, \sigma^*_{\mathtt{FIX}})$, with $\mathsf{m}^*_{\text{SIG}} = (\mathsf{m}^*_{\mathtt{FIX}}, \mathsf{ADM}^*, \mathbf{pk}^{(J^*)}_{\text{SAN}}, \mathbf{key}^*)$, comes from an oracle (related to the SIGN one). There are next two sub-cases that can be concluded similarly. It depends on whether the adversary wins its own experiment with both the conditions $I_{\text{ALG}} = \mathsf{sig}$ and $\text{JUDGE}(\mathsf{m}^*, \sigma^*, (\text{ALG}, I^*_{\text{ALG}}, \tau^*_{\text{ALG}}), \mathsf{PK}) = 1$ or with the condition $I_{\text{SIG}} = (\mathsf{sig}, \star)$ and $\text{JUDGE}(\mathsf{m}^*, \sigma^*, (\text{SIG}, I_{\text{SIG}}, \tau_{\text{SIG}}), \mathsf{PK}) = 1$. In fact, it does not matter since, in both cases, $\mathcal{B}_u$ can conclude as follows. As the signature of knowledge U'^* underlying $\sigma^*_{\mathtt{FULL}}$ is a secure ZK proof, we can extract, in the random oracle model, namely the forking lemma [19], the value $(\mathsf{B}_j, \mathsf{y}_j)$ which corresponds to the used certificate for the trapdoor or proof. As the adversary $\mathcal{A}$ wins with non-negligible probability, this certificate is related to a signer. (I.i) Either this certificate comes from the signing oracle of the double unforgeability experiment played by $\mathcal{B}_u$, and we cannot conclude and output $\perp$. (I.ii) Or it is a new certificate and $\mathcal{B}_u$ can send it back to its challenger to win the double unforgeability game. (II) $(\mathsf{m}^*_{\text{SIG}}, \sigma^*_{\mathtt{FIX}})$ is a new couple which has not been output by an oracle. In this case, we cannot conclude and output $\perp$.

One-more discrete logarithm. We finally interact with $\mathcal{A}$ to construct a machine $\mathcal{B}_o$ against the one-more discrete logarithm assumption. $\mathcal{B}_o$ first receives from its challenger the n values denoted $(\mathsf{v}_1, \ldots, \mathsf{v}_n)$ for which it has to compute the discrete logarithm in base b. From this, $\mathcal{B}_u$ executes the different key generation phases with the sole constraint that it should use the parameter b and the v_i's for the signer's public key (see the SIGKG phase during step 4 of the generation phase in Section 6.1). As we control $\mathcal{CA}$ during this phase, this is easily done. (i) If $\mathcal{A}_{\mathsf{ch}}$ makes a $\text{CORRUPT}(\mathsf{sig}, i, \cdot)$ call, $\mathcal{B}_o$ uses its own oracle to obtain the i-th discrete logarithm of its challenge. It next adds other keys related to this signer and gives the whole to the adversary. (ii) If $\mathcal{A}_{\mathsf{ch}}$ asks for a SIGN oracle, $\mathcal{B}_o$ first executes the step 2 of the SIGN procedure given in Section 6.2 to create the group member keys of all the sanitizers given by $\mathcal{A}_{\mathsf{ch}}$ plus the chosen signer. $\mathcal{B}_o$ next produces the proof of knowledge U as usual, using the keys it has previously generated. It next simulates the signature of knowledge U as it cannot produce it exactly since it does not necessarily know the underlying secret value u_i. The simulation is easily done, using standard technique (see *e.g.* [14]) in the random oracle model. (iii) Other oracles can be easily simulated. $\mathcal{A}_{\mathsf{ch}}$ eventually outputs a message m^* and a valid signature $\sigma^* = (\sigma^*_{\mathtt{FIX}}, \sigma^*_{\mathtt{FULL}}, \mathsf{ADM}^*, J^*, \mathsf{w}^*_i, \mathbf{key}^*)$, a honest signer i^* and, optionally, a couple $(I^*_{\text{ALG}}, \tau^*_{\text{ALG}})$ output by the ALGOPEN procedure. (I) $(\mathsf{m}^*_{\text{SIG}}, \sigma^*_{\mathtt{FIX}})$, with $\mathsf{m}^*_{\text{SIG}} = (\mathsf{m}^*_{\mathtt{FIX}}, \mathsf{ADM}^*, \mathbf{pk}^{(J^*)}_{\text{SAN}}, \mathbf{key}^*)$, comes from an oracle (related to the SIGN one). As for the double unforgeability case above, the signature of knowledge U'^* underlying $\sigma^*_{\mathtt{FULL}}$ is a secure ZK proof and we can consequently extract from U'^*, using standard techniques and in the random oracle model, the value $(\mathsf{B}_j, \mathsf{y}_j)$ which corresponds to the used certificate for the trapdoor or proof. As the adversary $\mathcal{A}$ wins with non-negligible probability the sanitizer accountability experiment, this certificate is related to a signer. (I.i) Either this certificate comes from the signing oracle. We next also extract from U'^* the underlying value u_j which, with non-negligible probability, does not come from the reveal oracle of the one-more discrete logarithm problem. $\mathcal{B}_o$ next asks its reveal oracle to obtain all remaining discrete logarithm (those of the

signers not been corrupted, except the one just obtained) and sends all n discrete logarithms to its challenger. This way, $\mathcal{B}_o$ solves the one-more discrete logarithm problem with non-negligible probability. (I.ii) Either this is a new certificate and we output $\perp$. (II) $(\mathsf{m}^*_{\text{SIG}}, \sigma^*_{\mathtt{FIX}})$ is a new couple which has not been output by an oracle. In this case, we cannot conclude and output $\perp$.
Conclusion. Consequently, as for the immutability and the traceability properties, it follows that we have $\mathbf{Succ}^{\mathsf{san\text{-}acc}}_{\Pi,\mathcal{A}}(\lambda, n, m) \leqslant \mathbf{Succ}^{\mathsf{trac}}_{\mathcal{GS},\mathcal{A}}(\lambda) + \mathbf{Succ}^{\mathsf{nf}}_{\mathcal{GS},\mathcal{A}}(\lambda) + \mathbf{Succ}^{\mathsf{2unf}}_{\mathcal{S},\mathcal{A}}(\lambda) + \mathbf{Succ}^{\mathsf{omdl}}_{\mathcal{A}}(\lambda)$.

Signer accountability. The output of an adversary against the signer's accountability, a message-signature pair and a couple (I^*, τ^*) which is related to either the AlgOpen or the SigOpen procedure, is accepted if the message-signature pair is valid (and does not come a call to the Sanitize oracle), if the given signature is not issued by a signer (who can always sanitize a message she has previously signed) and if one of the two below conditions is verified: (A) $I^* = \mathsf{san}$ and $\text{Judge}(\mathsf{m}^*, \sigma^*, (\text{ALG}, I^*, \tau^*), \mathsf{PK}) = 1$; (B) $I^* = (\mathsf{san}, 0)$ and $\text{Judge}(\mathsf{m}^*, \sigma^*, (\text{SIG}, I^*, \tau^*), \mathsf{PK}) = 1$. We show it contradicts either the double unforgeability of the $\mathcal{CA}$'s signature scheme with extensions denoted $\mathcal{S}$ or the one-more discrete logarithm assumption (see [4]). We define two different games we repeatedly play at random with an adversary $\mathcal{A}$ until one succeeds. These games are very close to the one related to the double unforgeability and the one-more discrete logarithm respectively and we do not repeat them again. The main difference is that both $\mathcal{B}_u$ and $\mathcal{B}_o$ need to switch the role of signers and sanitizers. We have $\mathbf{Succ}^{\mathsf{sig\text{-}acc}}_{\Pi,\mathcal{A}}(\lambda, n, m) \leqslant \mathbf{Succ}^{\mathsf{2unf}}_{\mathcal{S},\mathcal{A}}(\lambda) + \mathbf{Succ}^{\mathsf{omdl}}_{\mathcal{A}}(\lambda)$.

Full sanitizer anonymity. The adversary controls all the signers. It has also access to an oracle to corrupt a sanitizer i, i.e. to obtain the corresponding secret key $\mathsf{sk}_{\text{SAN}}[i] = (\mathsf{u}_i)$. At a given moment, the adversary chooses two honest sanitizers j_0^* and j_1^*, a (valid) message-signature pair (m, σ) and some modifications MOD. During the challenge phase, the sanitizer j_b^* modifies the pair (m, σ) with the modification MOD. The obtained pair is returned to the adversary. $\mathcal{A}$ returns a bit b^* and wins if $b^* = b$. We show it contradicts the IND-CCA of the Twin ElGamal scheme, the ZK property of the *trapdoor or proof* (regarding the certified secrets) or the perfect hiding property of the Pedersen commitment. The exchange between the adversary and the oracle are simulated by a distinguisher $\mathcal{D}$ which tries to break one of the previously cited properties.

Game 0. Game 0 corresponds to the experiment played by the adversary. Note that, the differences between a modification from j_0^* and a modification from j_1^* lie only in the $\sigma_{\mathtt{FULL}}$ part of the signature and, more precisely, in the certificates A_j and B_j. We return a valid modified pair from the honest sanitizer j_b^* using the corresponding certificates A_b and B_b for $\sigma_{\mathtt{FULL}}$. At the end of the game, the adversary returns a bit b^*. We note S_0 the event "$b = b^*$" in the Game 0. We get : $\mathbf{Adv}^{\mathsf{full\text{-}san\text{-}ano}}_{\mathcal{A}}(\lambda) = 2|Pr[S_0] - 1/2|$.

Game 1. $\mathcal{D}$ modifies its answers for the challenge. As U is a SoK obtained thanks to the Fiat-Shamir heuristic on a ZK PoK, we can simulate parts of this proof. In both cases, $b = 0$ and $b = 1$, $\mathcal{D}$ encrypts the certificate A_0. The $T_i's$ obtained are always an encryption of A_0. It must simulate the corresponding part of the proof U in order to hide it. A distinguisher between this game and the previous one can break the IND-CCA of the Twin Elgamal scheme: the encryption keys b, b' become the key obtained by the IND-CCA experiment, answers to the sanitization oracle are simulated thanks to calls to the encryption oracle and the two plaintexts chosen for the challenge step become A_0 and A_1. If we note S_1 the event "$b = b$*" in the Game 1, the difference between the two games is : $|Pr[S_1] - Pr[S_0]| \leqslant \mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{E},\mathcal{A}}(\lambda) + \mathbf{Adv}^{\mathsf{ZK}}_{\mathrm{POK},\mathcal{A}}(\lambda)$.

Game 2. $\mathcal{D}$ modifies the certificate B_b used in the case $b = 1$. In a similar way than in the previous game, the distinguisher modifies the certificate used in the *trapdoor or proof.* In both cases, it commits the same certificate B_0 in C_1. As C_1 is a Pedersen commitment, it is perfectly hiding. Thus it is impossible for the adversary to distinguish this modification. This implies to (also) simulate the last part of the proof U. As breaking the ZK property of the POK has already been considered in the previous game, the difference between this games and the previous one is : $|Pr[S_2] - Pr[S_1]| = \mathbf{Adv}^{\mathsf{hid}}_{PC,\mathcal{A}}(\lambda) = 0$. In that last game both cases are identical. Thus the probability of an adversary in that game is : $|Pr[S_2]| = 1/2$.

In conclusion, the advantage of an adversary against the full sanitizer anonymity of our scheme is : $\mathbf{Adv}^{\text{full-san-ano}}_{\mathcal{A}}(\lambda) = 2|Pr[S_0] - 1/2| = 2|Pr[S0] - Pr[S_2]| \leqslant 2(|Pr[S_1] - Pr[S_0]| + |Pr[S_2] - Pr[S_1]|) \leqslant 2\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{E},\mathcal{A}}(\lambda) + 2\mathbf{Adv}^{\mathsf{ZK}}_{\mathrm{POK},\mathcal{A}}(\lambda) + 2\mathbf{Adv}^{\mathsf{hid}}_{PC,\mathcal{A}}(\lambda) \leqslant 2\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{E},\mathcal{A}}(\lambda) + 2\mathbf{Adv}^{\mathsf{ZK}}_{\mathrm{POK},\mathcal{A}}(\lambda)$.

Full signer anonymity. This proof is similar to the previous one, on both the $\sigma_{\mathtt{FIX}}$ and $\sigma_{\mathtt{FULL}}$ group signatures. It only implies to add a game considering the modification of certificate A in $\sigma_{\mathtt{FIX}}$. As this modification is identical as the one seen in Game 1 in the previous proof, we do not supply more details. Note that the steps previously described hold because encryption keys belong to $\mathcal{O}_{\mathrm{ORI}}$ and $\mathcal{O}_{\mathrm{FULL}}$. Thus the corresponding public keys (b, b' and h, h') stay also under our control in that case. In consequence, if the adversary distinguish the game before the modification of the certificate A_b in one of the signature and after this modification, the distinguisher can use it to break the IND-CCA of the Twin ElGamal scheme : the encryption keys b, b' (for $\sigma_{\mathtt{FULL}}$) (resp. h, h' for $\sigma_{\mathtt{FIX}}$) are replaced by the key given by the IND-CCA experiment, answers to the sign oracle are simulated thanks to calls to the encryption oracle and the two plaintexts chosen for the challenge step become A_0 and A_1. In conclusion, the advantage of an adversary against the full signer anonymity of our scheme is : $\mathbf{Adv}^{\text{full-sig-ano}}_{\mathcal{A}}(\lambda) \leqslant 4\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{E},\mathcal{A}}(\lambda) + 4\mathbf{Adv}^{\mathsf{ZK}}_{\mathrm{POK},\mathcal{A}}(\lambda) + 2\mathbf{Adv}^{\mathsf{hid}}_{PC,\mathcal{A}}(\lambda) \leqslant 4\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{E},\mathcal{A}}(\lambda) + 4\mathbf{Adv}^{\mathsf{ZK}}_{\mathrm{POK},\mathcal{A}}(\lambda)$.

Non-frameability. The output of an adversary against the non-frameability property, a message-signature pair and a user-proof pair, is accepted if either the proof revealed that either the valid pair comes from an honest user i which did not produce this pair, or if the valid pair has the honest signer i as origin while the corresponding oracle has not been queried for this user. Note that the adversary may win this experiment in two cases, either $k = \text{ORI}$ or $k = \text{FULL}$. In the first case, this implies to consider the $\sigma_{\texttt{FIX}}$ signature. As this is exactly a group signature [14] on the message m_{SIG} and the proof τ_{ORI} exactly an opening proof of this signature, it is straightforward that an adversary against the non-frameability of our scheme can be used against the non-frameability of the group signature [14]. Thus we only consider the second case, i.e. $k = $FULL. We show that a successful adversary can be used to construct a machine $\mathcal{B}_o$ against the one-more discrete logarithm assumption. $\mathcal{B}_o$ first receives from its challenger the n values denoted $(\mathsf{v}_1, \ldots, \mathsf{v}_n)$ for which we inquire for the discrete logarithm in base b. From this, $\mathcal{B}_n$ executes the different key generation phases with the only constraint that it should use the parameter b and the v_i's for the user's public key (see the SigKG phase during step 4 of the generation phase in Section 6.1). Here, we use that $\mathcal{B}_n$ impersonates $\mathcal{CA}$. (i) If $\mathcal{A}$ requires a corruption request, $\mathcal{B}_o$ uses its own reveal oracle to obtain the i-th discrete logarithm of its challenge. For a signer corruption, it next adds the other keys related to this signer. (ii) If $\mathcal{A}$ asks for a Sign oracle, $\mathcal{B}_o$ first executes the step 2 of the Sign procedure given in Section 6.2 to create the group member keys of all the sanitizers given by $\mathcal{A}$ plus the chosen signer. $\mathcal{B}_o$ next produces the proof of knowledge U and simulates the signature of knowledge U. The simulation is easily done, using standard technique (see *e.g.* [14]) in the random oracle model. (iii) If $\mathcal{A}$ asks for a Sanitize oracle, $\mathcal{B}_o$ simulates the signature of knowledge U, as for the Sign request. $\mathcal{A}$ eventually outputs a message m^* and a valid signature $\sigma^* = (\sigma^*_{\texttt{FIX}}, \sigma^*_{\texttt{FULL}}, \mathsf{ADM}^*, J^*, \mathsf{w}^*_i, \mathbf{key}^*)$, a honest user i^* and a proof τ^*_{FULL} output by the FullOpen procedure. As $i \notin \mathcal{CU}$ the discrete logarithm of v_i has not been required to the challenger. The signature of knowledge U'^* underlying $\sigma^*_{\texttt{FULL}}$ is a secure ZK proof and we can consequently extract from U'^*, using standard techniques in the random oracle model, the value $s_i + \mathsf{u}_i$. Then s_i being a public data, we can recover u_i. $\mathcal{B}_o$ next asks its reveal oracle to obtain all remaining discrete logarithm (those of the users not been corrupted, except the one just obtained) and sends all n discrete logarithms to its challenger. This way, $\mathcal{B}_o$ solves the one-more discrete logarithm problem with non-negligible probability and we have : $\mathbf{Succ}^{\mathsf{nf}}_{\Pi,\mathcal{A}}(\lambda, n, m) \leqslant \mathbf{Succ}^{\mathsf{omdl}}_{\mathcal{A}}(\lambda)$.

Full transparency. Adversary has access to an oracle to corrupt sanitizers or signers and obtain the secret key $\mathbf{sk}_{\text{SAN}}[i] = (\mathsf{u}_i)$ or $\mathbf{sk}_{\text{SIG}}[i] = (y_i,\ \mathsf{u}_i,\ \alpha^{(i)}_1,\ \ldots,\ \alpha^{(i)}_\mu)$. At a given moment, the adversary chooses a honest signer i^* and a honest sanitizer j^*, a message m^*, admissible parts for this message ADM and some (valid) modifications MOD. During the challenge phase, the signer i signs the message m^* considering ADM. The sanitizer j modifies the obtained pair (m, σ) with the modification MOD and, if $b = 0$, this modified pair is returned to the adversary. If $b = 1$ the signer i signed the message m'^* modified according to MOD and this pair is returned to the adversary. $\mathcal{A}$ returns a bit b^* and win if $b^* = b$. We show it contradicts the IND-CCA of the Twin ElGamal scheme, the ZK property of the *trapdoor or proof* (regarding the certified secrets) or the perfectly hiding property of the Pedersen commitment. Messages between the adversary and the oracle are simulated by a distinguisher $\mathcal{D}$ which tries to break one of those properties.

Game 0. This is the experiment played by the adversary. At the end of the game, the adversary returns a bit b^*. We note S_0 the event "$b = b^*$" in the Game 0. We get : $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{full\text{-}trans}}(\lambda) = 2|Pr[S_0] - 1/2|$.

Game 1. Note that the difference between an original from i and a modification from j lies on the signature $\sigma_{\mathtt{FULL}}$ and, especially, on the A_j and B_i certificate used in that signature. In this game, $\mathcal{D}$ modifies its answers in $\sigma_{\mathtt{FULL}}$. In both case $b = 0$ and $b = 1$, it encrypts the certificate A of the signer i. Thus, both signatures use the encryption of the same certificate A. In consequence, when $b = 1$ we must simulate the corresponding part of the proof U, which relies on the ZK property of Pok. If this changes the success probability of the adversary in a significant amount, the distinguisher can use it to break the IND-CCA of the Twin Elgamal scheme. $\mathcal{D}$ generate a valid pair of keys for $\mathcal{O}_{\text{ORI}}$ and publishes h, h' but it uses the public key obtained by the IND-CCA experiment for $\mathcal{O}_{\text{FULL}}$ public keys b, b'. It can answer requests to oracles sign and sanitize by using the algorithm for the $\sigma_{\mathtt{FIX}}$ signature and the encryption oracle with a simulated proof U for $\sigma_{\mathtt{FULL}}$. Finally, the two plaintexts chosen for the challenge step become A and A_j. If we note S_1 the event "$b = b^*$" in the Game 1, the difference between the two games is : $|Pr[S_1] - Pr[S_0]| \leqslant \mathbf{Adv}_{\mathcal{E},\mathcal{A}}^{\mathsf{IND\text{-}CCA}}(\lambda) + \mathbf{Adv}_{\text{Pok},\mathcal{A}}^{\mathsf{ZK}}(\lambda)$.

Game 2. $\mathcal{D}$ modifies the certificate B_j used in the case $b = 1$. As certificates are generated differently between a signer and a sanitizer, the distinguisher modifies the certificate used in the *trapdoor or proof.* If $b = 1$, it computes a new certificate $B'_j = (g_{\text{SIG}} b^{u_j})^{\frac{1}{\gamma + y_j}}$ to commit and prove. A distinguisher between this game and the previous one can break the ZK property of the *trapdoor or proof.* Thus: $|Pr[S_2] - Pr[S_1]| \leqslant \mathbf{Adv}_{trap\text{-}or,\mathcal{A}}^{\mathsf{ZK}}(\lambda)$.

Game 3. Finally, $\mathcal{D}$ modifies the certificate B_b used in the case $b = 1$. It will always use the certificate B_i of the signer. In both cases, it commits to the same certificate B_i in C_1. As C_1 is a Pedersen commitment, it is perfectly hiding. Thus it is impossible for the adversary to distinguish this modification. It implies to simulate the last part of the proof U as well. As breaking the ZK property of the Pok has already been considered in the previous game, the difference between this game and the previous one is : $|Pr[S_3] - Pr[S_2]| = \mathbf{Adv}_{PC,\mathcal{A}}^{\mathsf{hid}}(\lambda) = 0$. In that last game both cases are identical. Thus the probability of an adversary in that game is : $|Pr[S_3]| = 1/2$.

In conclusion, the advantage of an adversary against the group transparency of our scheme is :
$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{full\text{-}trans}}(\lambda) = 2|Pr[S_0] - 1/2| = 2|Pr[S0] - Pr[S_3]| \leqslant 2(|Pr[S_1] - Pr[S_0]| + |Pr[S_2] - Pr[S_1]| + |Pr[S_3] - Pr[S_2]|) \leqslant 2\mathbf{Adv}_{\mathcal{E},\mathcal{A}}^{\mathsf{IND\text{-}CCA}}(\lambda) + 2\mathbf{Adv}_{\text{Pok},\mathcal{A}}^{\mathsf{ZK}}(\lambda) + 2\mathbf{Adv}_{td,\mathcal{A}}^{\mathsf{ZK}}(\lambda) + 2\mathbf{Adv}_{PC,\mathcal{A}}^{\mathsf{hid}} \leqslant 2\mathbf{Adv}_{\mathcal{E},\mathcal{A}}^{\mathsf{IND\text{-}CCA}}(\lambda) + 2\mathbf{Adv}_{\text{Pok},\mathcal{A}}^{\mathsf{ZK}}(\lambda) + 2\mathbf{Adv}_{td,\mathcal{A}}^{\mathsf{ZK}}(\lambda)$.