Unlinkable and Strongly Accountable Sanitizable Signatures from Verifiable Ring Signatures*

Xavier Bultel[1,2] and Pascal Lafourcade[1,2]

[1] CNRS, UMR 6158, LIMOS, F-63173 Aubière, France
[2] Université Clermont Auvergne, BP 10448, 63000 Clermont-Ferrand, France

Abstract. An *Unlinkable Sanitizable Signature* scheme (USS) allows a sanitizer to modify some parts of a signed message such that nobody can link the modified signature to the original one. A *Verifiable Ring Signature* scheme (VRS) allows the users to sign messages anonymously within a group such that a user can prove *a posteriori* to a verifier that he is the signer of a given message. In this paper, we first revisit the notion of VRS: we improve the proof capabilities of the users, we give a complete security model for VRS and we give an efficient and secure scheme called EVeR. Our main contribution is GUSS, a generic USS based on a VRS scheme and an unforgeable signature scheme. We show that GUSS instanciated with EVeR and the Schnorr's signature is twice as efficient as the best USS scheme of the literature. Moreover, we propose a stronger definition of *accountability*: an USS is *accountable* when the signer can prove whether a signature is sanitized. We formally define the notion of *strong accontability* when the sanitizer can also prove the origin of a signature. We show that the notion of strong accountability is important in practice. Finally, we prove the security properties of GUSS (including the strong accountability) and EVeR under the Decisional Diffie-Hellman assumption in the random oracle model.

1 Introduction

Sanitizable Signatures (SS) were introduced by Ateniese *et al.* [1], but similar primitives were independently proposed in [23]. In this primitive, a *signer* allows a proxy (called the *sanitizer*) to modify some parts of a signed message. For example, a magistrate wishes to delegate the power to summon someone to the court to his secretary. He signs the message "*Franz* is summoned to court for an interrogation on *Monday*" and gives the signature to his secretary, where "Franz" and "Monday" are sanitizable and the other parts are fixed. Thus, in order to summoned Joseph K. on Saturday in the name of the magistrate, the secretary can change the signed message into "*Joseph K.* is summoned to the court for an interrogation on *Saturday*".

Ateniese *et al.* in [1] propose some applications of this primitive in privacy of health data, authenticated media streams and reliable routing informations. They also introduced five security properties formalized by Brzuska *et al.* in [4]:
Unfogeability: no unauthorised user can generate a valid signature.

* This research was conducted with the support of the "Digital Trust" Chair from the University of Auvergne Foundation.

Immutability: sanitizer cannot transform a signature from an unauthorised message.
Privacy: no information about the original message is leaked by a sanitized signature.
Transparency: nobody can say if a signature is sanitized or not.
Accountability: the signer can prove that a signature is sanitized or is the original one.

Finally, in [6] authors point a non-studied but relevant property called *unlinkability*: a scheme is unlinkable when it is not possible to link a sanitized signature to the original one. The authors give a generic unlinkable scheme based on group signatures. In 2016, Fleischhacker *et al.* [16] give a more efficient construction based on signatures with re-randomizable keys.

On the other hand, ring signature is a well-studied cryptographic primitive introduced by Shamir *et al.* in [22] where any user can sign anonymously within an ad-hoc group of users. Such a scheme is *verifiable* [21] when any user can prove *a posteriori* to a verifier that he is the signer of a given message. In this paper, we improve the proof properties of VRS, we give an efficient VRS scheme called EVeR and a generic unlinkable sanitizable signature scheme called GUSS that uses verifiable ring signatures. We also show that the definition of accountability is too weak for practical uses, and we propose a stronger definition.

Contributions: Existing VRS schemes allow any user to prove that he is the signer of a given message. We extend the definition of VRS to allow a user to prove that he is not the signer of a given message. We give a formal security model for VRS that takes into account this property. We first extend the classical security properties of ring signatures to verifiable ring signatures, namely the *unforgeability* (no unauthorised user can forge a valid signature) and the *anonimity* (nobody can distinguish the signer in the group). In addition we define the *accountability* (a user cannot sign a message and prove that he is not the signer) and the *non-usurpability* (a user cannot prove that he is the signer of a message if it is not true, and a user cannot forge a message such that the other users cannot prove that they are not the signers). To the best of our knowledge, it is the first time that formal security models are proposed for VRS. We also design an efficient secure VRS scheme under the decisional Diffie-Hellman assumption in the random oracle model.

The usual definition of accountability for SS considers that the signer can prove the origin of a signature (signer or sanitizer) using a proof algorithm such that:

1. The signer cannot forge a signature together with a proof that the signature comes from the sanitizer.
2. The sannitizer cannot forge a signature such that the proof algorithm accuses the signer.

The proof algorithm requires the secret key of the signer. To show that this definition is too weak, we consider a dishonest signer who refuses to proof the origin of a litigious signature. The dishonest signer claims that he lost his secret key because of problems with his hard drive. There is no way to verify whether the signer is lying. Unfortunately, without his secret key, the signer cannot generate the proof for the litigious signature. Then nobody can judge if the signature is sanitized or not and there is a risk of accusing the honest sanitizer wrongly. To solve this problem, we add a second proof algorithm that allows the sanitizer to prove the origin of a signature. To achieve the strong accountability, the two following additional properties are required:

1. The sanitizer cannot sanitize a signature σ and prove that σ is not sanitized.
2. The signer cannot forge a signature such that the sanitizer proof algorithm accuses the sanitizer.

The main contribution of this paper is to propose an efficient and generic unlinkable SS scheme called GUSS. This scheme is instantiated by a VRS and an unforgeable signature scheme. It is the first SS scheme that achieves strong accountability. We compare GUSS with the other schemes of the litterature:

Brzuska *et al.* [6] This scheme is based on group signatures. Our scheme is build on the same model, but it uses ring signatures instead of group signatures. The main advantage of group signatures is that the size of the signature is not proportional to the size of the group. However, for small groups, ring signatures are much more efficient than group signatures. Since the scheme of Brzuska *et al.* and GUSS uses group/ring signatures for groups of two users, GUSS is much more practicale for an equivalent level of genericity.

Fleischhacker *et al.* [16] This scheme is based on *signatures with re-randomizable keys*. It is generic, however it uses different tools that must have special properties to be compatible with each other. To the best of our knowledge, it is the most efficient scheme of the literature. GUSS instantiated with EVeR and the Schnorr's signature is twice as efficient as the best instantiation of this scheme. In Fig. 1, we compare the efficiency of each algorithm of our scheme and the scheme of Fleischhacker *et al.*.

Lai *et al.* [19] Recently, Lai *et al.* proposed an USS that is secure in the standard model, however it uses pairing and it is much less efficient than the scheme of Fleischhacker *et al.* that is in the random oracle model, thus it is much less efficient than our scheme. In their paper [19], Lai *et al.* give a comparaison of the efficiency of the three schemes of the litterature.

	SiGen	SaGen	Sig	San	Ver	SiProof	SiJudge	**Total**	pk	spk	sk	ssk	σ	π
[16]	7	1	15	14	17	23	6	**73**	7	1	14	1	14	4
GUSS	2	1	8	7	10	3	2	**36**	2	1	2	1	12	4

Fig. 1. Comparison of GUSS and the scheme of Fleischhacker *et al.*: The first six columns give the number of exponentiations of each algorithms of both schemes, namely the key generation algorithm of the signer (SiGen) and the sanitizer (SaGen), the signature algorithm (Sig), the verification algorithm (Ver), the sanitize algorithm (San), the proof algorithm (SiProof) and the judge algorithm (SiJudge). The last six columns gives respectively the size of the public key of the signer (pk) and the sanitizer (pk), the size of the secret key of the signer (sk) and the sanitizer (ssk), the size of a signature (σ) and the size of a proof (π) outputted by SiProof. This size is measured in elements of a group G of prime order. As in [16], for the sake of clarity, we do not distinguish between elements of G and elements of $\mathbb{Z}_p^*$. We consider the best instantiation the scheme of Fleischhacker *et al.* given in [16].

Related works: *Sanitizable Signatures* (SS) was first introduced by Ateniese *et al.* [1]. Later, Brzuska *et al.* give formal security definitions [5] for *unfogeability*, *immutability*, *privacy*, *transparency* and *accountability*. *Unlinkability* was introduced and

formally defined by Brzuska *et al.* in [6]. In [7], Brzuska *et al.* introduce an alternative definition of accountability called *non-interactive public accountability* where the capability to prove the origin of a signature is given to a third party. One year later, the same authors propose a stronger definition of unlinkability [8] and design a scheme that is both strongly unlinkable and non-interactive public accountable. However, non-interactive public accountability is not compatible with transparency. In this paper, we focus on schemes that are unlinkable, transparent and interactive accountable. To the best of our knowledge, there are only 3 schemes with these 3 properties, *i.e.* [6, 16, 19].

Some works are focused on other properties of SS that we do not consider here, as SS with multiple sanitizers [10], or SS where the power of the sanitizer is limited [9]. Finally, there exist other primitives that solve related but different problems as homomorphioc signatures [18], redactable signatures [3] or proxy signatures [17]. Differences between these primitives and sanitizable signatures are detailed in [16].

On the other hand, *Ring Signatures* (RS) [22] were introduced by rivest *et al.* in 2003 and *Verifiable Ring Signatures* (VRS) [21] were introduced in 2003 by Lv. RS allows the users to sign anonymously within a group, and VRS allows a user to prove that he is the signer of a given message. To the best of our knowledge, even if several VRS have been proposed [12, 24], there is no security model for this primitive in the litterature. Convertible ring signatures [20] are very closed to verifiable ring signatures, it allows the signer of an anonymous (ring) signature to transform it into a standard signature (*i.e.* a *desanonimized* signature). It can be used as a verifiable ring signature because the desanonimized signature can be viewed as a proof that the user is the signer of a given message. However, in this paper we propose a stronger definition of VRS where a user also can prove that he is not the signer of a message, and this property cannot be acheived using convertible signatures.

A *List Signature* scheme (LS) [11] is a kind of RS that have the following property: if a user signs two messages for the same *event-id*, then it is possible to link these signatures and the user identity is publicly revealed. It can be used to design a VRS in our model: to prove whether he is the signer of a given message, the user signs a second message using the same event-id. If the two signatures are linked, then the judge is convinced that the user is the signer, else he is convinced that the user is not the signer. However, LS requires security properties that are too strong for VRS (linkability and traceability) and it would result in less efficient schemes.

Outline: In section 2, we present the formal definition and the security models for both verifiable ring signatures and unlinkable sanitizable signatures. In section 3, we present our two schemes EvER and GUSS, before concluding in section 4. Moreover, we recall in appendix A the standard cryptographic definitions used in this paper, namely the DDH assumption, the deterministic digital signatures (DS), the Schnorr's signature and the non-interactive zero-knowledge proofs (NIZKP).

2 Formal Definitions

2.1 Verifiable Ring Signatures

We give formal definitions and secuirty of Verifiable Ring Signatures (VRS). A VRS is a ring signature scheme where a user can prove to a judge if he is the signer of a message or not. It is composed of 6 algorithms. V.Init, V.Gen, V.Sig and V.Ver are define as in the usual ring signature definitions. V.Gen generates public and private keys. V.Sig anonymously signs a message according to a set of public keys. V.Ver verifies the soundness of a signature. A VRS has two additional algorithms: V.Proof allows a user to prove whether he is the signer of a message or not, and V.Judge allows to verify the proofs outputted by V.Proof.

Definition 1 (Verifiable Ring Signature (VRS)). *A* Verifiable Ring Signature *scheme is a tuple of* 6 *algorithms defined by:*

V.Init(1^k): *It returns a setup value* init.

V.Gen(init): *It returns a pair of signer public/private keys* (pk, sk).

V.Sig(L, m, sk): *This algorithm computes a signature* σ *using the key* sk *for the message* m *according to the set of public keys* L.

V.Ver(L, m, σ): *It returns a bit* b*: if the signature* σ *of* m *is valid according to the set of public key* L *then* $b = 1$*, else* $b = 0$.

V.Proof$(L, m, \sigma, \mathsf{pk}, \mathsf{sk})$: *It returns a proof* π *for the signature* σ *of* m *according to the set of public key* L.

V.Judge$(L, m, \sigma, \mathsf{pk}, \pi)$: *It returns a bit* b *or the bottom symbol* $\perp$*: if* $b = 1$ *(resp.* 0*) then* π *proves that* σ *comes from (resp. does not come from) the signer corresponding to the public key* pk*. It outputs* $\perp$ *when the proof is not well formed.*

Unforgeability: We first adapt the unforgeability property of ring signatures to VRS. Informally, a VRS is unforgeable when no adversary is able to forge a signature for a ring of public keys without any corresponding secret key. In this model, the adversary has access to a signature oracle V.Sig$(\cdot, \cdot, \cdot)$ (that outputs signatures of chosen messages for chosen users in the ring) and a proof oracle V.Proof$(\cdot, \cdot, \cdot, \cdot, \cdot)$ (that computes proofs as the algorithm V.Proof for chosen signatures and chosen users). The adversary successes the attack when he outputs a valid signature that was not already computed by the signature oracle.

Definition 2 (Unforgeability). *Let* P *be a VRS of security parameter* k*,* n *be an integer. We consider two oracles:*

V.Sig$(\cdot, \cdot, \cdot)$**:** *On input* (L, l, m)*, if* $1 \leq l \leq n$ *then this oracle returns the message* V.Sig(L, sk_l, m)*, else it returns* $\perp$.

V.Proof$(\cdot, \cdot, \cdot, \cdot, \cdot)$**:** *On input* (L, m, σ, l)*, if* $1 \leq l \leq n$ *then this proof oracle returns* V.Proof$(L, m, \sigma, \mathsf{pk}_l, \mathsf{sk}_l)$*, else it returns* $\perp$.

P *is* n-unf *secure when for any polynomial time adversary* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *wins the following experiment is negligible, where* q_S *is the number of calls to the oracle* V.Sig$(\cdot, \cdot, \cdot)$ *and* σ_i *is is the* i^{th} *signature outputted by this oracle:*

$\mathbf{Exp}^{n\text{-}\mathbf{unf}}_{P,\mathcal{A}}(k)$:
$\mathit{init} \leftarrow \mathit{V.Init}(1^k)$
$\forall 1 \leq i \leq n, (\mathit{pk}_i, \mathit{sk}_i) \leftarrow \mathit{V.Gen}(\mathit{init})$
$(L_*, \sigma_*, m_*) \leftarrow \mathcal{A}^{\mathit{V.Sig}(\cdot,\cdot,\cdot),\mathit{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\{\mathit{pk}_i\}_{1\leq i\leq n})$
if $\mathit{V.Ver}(L_*, \sigma_*, m_*) = 1$ *and* $L_* \subseteq \{\mathit{pk}_i\}_{1\leq i\leq n}$ *and* $\forall\, i \in \{1, \ldots, q_S\}$, $\sigma_i \neq \sigma_*$
then return 1, *else return* 0

P *is* unforgeable *when it is* n-unf *secure for any polynomialy bounded* n.

Annonymity: we adapt the anonymity property of ring signatures to VRS. Informally, a VRS is anonymous when no adversary is able to link a signature to the corresponding user. The adversary has access to the signature oracle and the proof oracle. During a first phase, he chooses two honest users in the ring, and in the second phase, he has access to a challenge oracle $\mathsf{LRSO}_\mathsf{b}(d_0, d_1, \cdot, \cdot)$ that outputs signatures of chosen messages using the secret key of one of the two chosen users. The adversary successes the attack if he guesses who is the user chosen by the challenge oracle. Note that the adversary cannot use the proof oracle on the signatures outputted by the challenge oracle.

Definition 3 (Anonymity). *Let* P *be a VRS of security parameter* k*, let* n *be an integer. Let the following oracle be:*
$\mathsf{LRSO}_\mathsf{b}(d_0, d_1, \cdot, \cdot)$**:** *On input* (m, L)*, if* $\{\mathit{pk}_{d_0}, \mathit{pk}_{d_1}\} \subseteq L$ *then this oracle returns* $\mathit{V.Sig}(L, \mathit{sk}_{d_b}, m)$*, else it returns* $\perp$.
P *is* n-ano *secure when for any polynomial time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the difference between* $1/2$ *and the probability that* $\mathcal{A}$ *wins the following experiment is negligible, where* $\mathit{V.Sig}(\cdot,\cdot,\cdot)$ *and* $\mathit{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$ *are defined as in Def. 2 and where* q_S *(resp.* q_P*) is the number of calls to the oracle* $\mathit{V.Sig}(\cdot,\cdot,\cdot)$ *(resp.* $\mathit{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$*),* $(L_i, m_i, \sigma_i, l_i)$ *is is the* i^{th} *query sent to oracle* $\mathit{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$ *and* σ'_j *is is the* j^{th} *signature outputted by the oracle* $\mathsf{LRSO}_\mathsf{b}(d_0, d_1, \cdot, \cdot)$*:*

$\mathbf{Exp}^{n\text{-}\mathbf{ano}}_{P,\mathcal{A}}(k)$:
$\mathit{init} \leftarrow \mathit{V.Init}(1^k)$
$\forall 1 \leq i \leq n, (\mathit{pk}_i, \mathit{sk}_i) \leftarrow \mathit{V.Gen}(\mathit{init})$
$(d_0, d_1) \leftarrow \mathcal{A}_1^{\mathit{V.Sig}(\cdot,\cdot,\cdot),\mathit{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\{\mathit{pk}_i\}_{1\leq i\leq n})$
$b \xleftarrow{\$} \{0, 1\}$
$b_* \leftarrow \mathcal{A}_2^{\mathit{V.Sig}(\cdot,\cdot,\cdot),\mathit{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot),\mathsf{LRSO}_\mathsf{b}(d_0,d_1,\cdot,\cdot)}(\{\mathit{pk}_i\}_{1\leq i\leq n})$
if $(b = b_*)$ *and* $(\forall\, i, j \in \{1, \ldots, \max(q_S, q_P)\}, (\sigma_i \neq \sigma'_j)$ *or* $(l_i \neq d_0$ *and* $l_i \neq d_1))$
then return 1, *else return* 0

P *is* anonymous *when it is* n-ano *secure for any polynomialy bounded* n.

Accounatbility: We consider an adversary that has access to a proof oracle and a signature oracle. A VRS is accountable when no adversary is able to forge a signature σ (that does not be outputted by the signature oracle) toghether with a proof that he is not the signer of σ. Note that the ring of σ must contain at most one public key that does not come from an honest user, thus the adersary knows at most one secret key that corresponds to a public key in the ring.

Definition 4 (Accountability). *Let* P *be a VRS of security parameter* k *and let* n *be an integer.* P *is* n-acc *secure when for any polynomial time adversary* $\mathcal{A}$*, the*

probability that $\mathcal{A}$ wins the following experiment is negligible, where $\mathsf{V.Sig}(\cdot,\cdot,\cdot)$ *and* $\mathsf{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$ *are defined as in Def. 2 and where q_S is the number of calls to the oracle* $\mathsf{V.Sig}(\cdot,\cdot,\cdot)$ *and σ_i is is the i^{th} signature outputted by this oracle:*

$\mathbf{Exp}^{n\text{-}\mathsf{acc}}_{P,\mathcal{A}}(k)$*:*
$\mathit{init} \leftarrow \mathsf{V.Init}(1^k)$
$\forall 1 \leq i \leq n, (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{V.Gen}(\mathit{init})$
$(L_*, m_*, \sigma_*, \mathsf{pk}_*, \pi_*) \leftarrow \mathcal{A}^{\mathsf{V.Sig}(\cdot,\cdot,\cdot),\mathsf{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\{\mathsf{pk}_i\}_{1\leq i\leq n})$
if $(L \subseteq \{\mathsf{pk}_i\}_{1\leq i\leq n} \cup \{\mathsf{pk}_*\})$ *and* $(\mathsf{V.Ver}(L_*, \sigma_*, m_*) = 1)$ *and*
$(\mathsf{V.Judge}(L_*, m_*, \sigma_*, \mathsf{pk}_*, \pi_*) = 0)$ *and* $(\forall\, i \in \{1, \ldots, q_S\},\, \sigma_i \neq \sigma_*)$
then return 1*, else return* 0

P *is* accountable *when it is n-*acc *secure for any polynomialy bounded n.*

Non-usurpability: We distinguish two experiments for this property:

- The first experiment, denoted non-usu-1, considers an adversary that has access to a proof oracle and a signature oracle. Its goal is to forge a valid signature with a proof that the signer is another user in the ring. Since this property is not required to build our generic USS, we give the formal definition of this security experiment in Appendix B.
- The second experiment, denoted non-usu-2, considers an adversary that has access to a proof oracle and a signature oracle and that recieves the public key of an honest user as input. The goal of the adversary is to forge a signature σ such that the proof algorithm runs by the honest user returns a proof that σ was computed by the honest user (*i.e.* the proof algorithm returns 1) or a non-valid proof (*i.e.* the proof algorithm returns $\perp$). Moreover, the signature σ must not come from the signature orale.

Definition 5 (Non-usurpability). *Let P be a VRS of security parameter k and let n be an integer. P is n-*$\mathsf{non\text{-}usu\text{-}2}$ *secure when for any polynomial time adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following experiment is negligible, where* $\mathsf{V.Sig}(\cdot,\cdot,\cdot)$ *and* $\mathsf{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$ *are defined as in Def. 2 and where q_S is the number of calls to the oracle* $\mathsf{V.Sig}(\cdot,\cdot,\cdot)$ *and σ_i is is the i^{th} signature outputted by this oracle:*

$\mathbf{Exp}^{n\text{-}\mathsf{non\text{-}usu\text{-}2}}_{P,\mathcal{A}}(k)$*:*
$\mathit{init} \leftarrow \mathsf{V.Init}(1^k)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{V.Gen}(\mathit{init})$
$(L_*, m_*, \sigma_*) \leftarrow \mathcal{A}^{\mathsf{V.Sig}(\cdot,\cdot,\cdot),\mathsf{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\mathsf{pk})$
$\pi \leftarrow \mathsf{V.Proof}(L_*, m_*, \sigma_*, \mathsf{pk}, \mathsf{sk})$
if $(\mathsf{V.Ver}(L_*, \sigma_*, m_*) = 1)$ *and*
$(\mathsf{V.Judge}(L_*, m_*, \sigma_*, \mathsf{pk}_*, \pi_*) \neq 0)$ *and* $(\forall\, i \in \{1, \ldots, q_S\},\, \sigma_i \neq \sigma_*)$
then return 1*, else return* 0

P *is* non-usurpable *when it is both n-*$\mathsf{non\text{-}usu\text{-}1}$ *(see Appendix. B) and n-*$\mathsf{non\text{-}usu\text{-}2}$ *secure for any polynomialy bounded n.*

2.2 Sanitizable Signature

We give the formal definition and security properties of the sanitizable signature primitive. Comparing to the previous definitions where only the signer can prove the origin

of a signature, our definition introduces algorithms that allow the sanitizer to prove the origin of a signature. Moreover, in addition to the usual security models of [5], we present two new security experiments that improve the accountability definition.

A SS scheme contains 10 algorithms. Init outputs the setup values. SiGen and SaGen generate respectively the signer and the sanitizer public/private keys. As in classical signature schemes, the algorithms Sig and Ver allow the users to sign a message and to verify a signature. However, signatures are computed using a sanitizer public key and an admissible function ADM. The algorithm San allows the sanitizer to transform a signature of a message m according to a modification function MOD: if MOD is admissible according to the admissible function (*i.e.* $\text{MOD}(\text{ADM}) = 1$) this algorithm returns a signature of the message $\text{MOD}(m)$.

SiProof allows the signer to prove whether a signature is sanitized or not. Proofs outpoutted by this algorithm can be checked by anybody using the algorithm SiJudge. Finally, algorithms SaProof and SaJudge have the same functionalities as SiProof and SiJudge, but the proof are computed from the secret parameters of the sanitizer instead of the signer.

Definition 6 (Sanitizable Signature (SS)). *A* Sanitizable Signature *scheme is a tuple of* 10 *algorithms defined as follows:*

$\mathsf{Init}(1^k)$: *It returns a setup value* init.
$\mathsf{SiGen}(\mathsf{init})$: *It returns a pair of signer public/private keys* (pk, sk).
$\mathsf{SaGen}(\mathsf{init})$: *It returns a pair of sanitizer public/private keys* (spk, ssk).
$\mathsf{Sig}(m, \mathsf{sk}, \mathsf{spk}, \text{ADM})$: *This algorithm computes a signature* σ *using the key* sk *for the message* m*, the sanitizer key* spk *and the admissible function* ADM*. Note that we assume that* ADM *can be efficiently recovered from any signature.*
$\mathsf{San}(m, \text{MOD}, \sigma, \mathsf{pk}, \mathsf{ssk})$: *Let the admissible function* ADM *according to the signature* σ*. If* $\text{ADM}(\text{MOD}) = 1$ *then this algorithm returns a signature* σ' *of the message* $m' = \text{MOD}(m)$ *using the signature* σ*, the signer public key* pk *and the sanitizer secret key* ssk*. Else it returns* $\perp$.
$\mathsf{Ver}(m, \sigma, \mathsf{pk}, \mathsf{spk})$: *It returns a bit* b*: if the signature* σ *of* m *is valid for the two public keys* pk *and* spk *then* $b = 1$*, else* $b = 0$.
$\mathsf{SiProof}(\mathsf{sk}, m, \sigma, \mathsf{spk})$: *It returns a signer proof* π_{si} *for the signature* σ *of* m *using the signer secret key* sk *and the sanitizer public key* spk.
$\mathsf{SaProof}(\mathsf{ssk}, m, \sigma, \mathsf{pk})$: *It returns a sanitizer proof* π_{sa} *for the signature* σ *of* m *using the sanitizer secret key* ssk *and the signer public key* pk.
$\mathsf{SiJudge}(m, \sigma, \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{si}})$: *It returns a bit* d *or the bottom symbol* $\perp$*: if* π_{si} *proves that* σ *comes from the signer corresponding to the public key* pk *then* $d = 1$*, else if* π_{si} *proves that* σ *comes from the sanitizer corresponding to the public key* spk *then* $d = 0$*, else the algorithm outputs* $\perp$.
$\mathsf{SaJudge}(m, \sigma, \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{sa}})$: *It returns a bit* d *or the bottom symbol* $\perp$*: if* π_{sa} *proves that* σ *comes from the signer corresponding to the public key* pk *then* $d = 1$*, else if* π_{sa} *proves that* σ *comes from the sanitizer corresponding to the public key* spk *then* $d = 0$*, else the algorithm outputs* $\perp$.

As it is mentioned in Introduction, SS schemes have the following security properties: *unfogeability*, *immutability*, *privacy*, *transparency* and *accountability*. In [5] au-

thors show that if a scheme has the *immutability*, the *transparency* and the *accountability* properties, then it has the *unforgeability* and the *privacy* properties. Hence we do not need to prove these two properties, then we do not recall their formal definitions.

Immutability: A SS is immutable when no adversary is able to sanitize a signature without the corresponding sanitizer secret key or to sanitize a signature using a modification function that is not admissible (*i.e.* $\text{ADM}(\text{MOD}) = 0$). To help him, the adversary has access to a signature oracle $\mathsf{Sig}(.,\mathsf{sk},.,.)$ and a proof oracle $\mathsf{SiProof}(\mathsf{sk},.,.,.)$.

Definition 7 (Immutability). *We consider the two following oracles:*
$\mathsf{Sig}(.,\mathsf{sk},.,.)$**:** *On input* $(m, \text{ADM}, \mathit{spk})$*, this oracle returns* $\mathit{Sig}(m, \mathit{sk}, \text{ADM}, \mathit{spk})$.
$\mathsf{SiProof}(\mathsf{sk},.,.,.)$**:** *On input* $(m, \sigma, \mathit{spk})$*, this oracle returns* $\mathit{SiProof}(\mathit{sk}, m, \sigma, \mathit{spk})$.
Let P *be a* SS *of security parameter* k*.* P *is* Immut *secure (or* immutable*) when for any polynomial time adversary* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *wins the following experiment is negligible, where* q_{Sig} *is the number of calls to the oracle* $\mathit{Sig}(.,\mathit{sk},.,.)$*,* $(m_i, \text{ADM}_i, \mathit{spk}_i)$ *is the* i^{th} *query asked to this oracle and* σ_i *is the corresponding response:*

$\mathbf{Exp}^{\mathsf{Immut}}_{P,\mathcal{A}}(k)$*:*
$\mathit{init} \leftarrow \mathit{Init}(1^k)$
$(\mathit{pk}, \mathit{sk}) \leftarrow \mathit{SiGen}(\mathit{init})$
$(\mathit{spk}_*, m_*, \sigma_*) \leftarrow \mathcal{A}^{\mathit{Sig}(.,\mathit{sk},.,.),\mathit{SiProof}(\mathit{sk},.,.,.)}(\mathit{pk})$
if $(\mathit{Ver}(m_*, \sigma_*, \mathit{pk}, \mathit{spk}_*) = 1)$ *and* $(\forall\, i \in \{1, \ldots, q_{\mathit{Sig}}\}, (\mathit{spk}_* \neq \mathit{spk}_i)$ *or*
$(\forall\, \text{MOD}$ *such that* $\text{ADM}_i(\text{MOD}) = 1, m_* \neq \text{MOD}(m_i)))$
then return 1*, else return* 0

Transparency: The transparency property guarantees that no adversary is able to distinguish whether a signature is sanitized or not. In addition to the signature oracle and the signer proof oracle, the adversary has access to a sanitize oracle $\mathsf{San}(.,.,.,.,\mathsf{ssk})$ that sanitizes chosen signatures and a sanitizer proof oracle $\mathsf{SaProof}(\mathsf{ssk},.,.,.)$ that computes sanitizer proofs for given signatures. Moreover the adversary has access to a challenge oracle $\mathsf{Sa/Si}(b,\mathsf{pk},\mathsf{spk},\mathsf{sk},\mathsf{ssk},.,.,.)$ that depends to a randomly chosen bit b: this oracle signs a given message and sanitizes it, if $b = 0$ then it outputs the original signature, else it outputs the sanitized signature. The adversary cannot use the proof oracles on the signatures outputted by the challenge oracle. To success the experiment, the adversary must guess b.

Definition 8 (Transparency). *We consider the following oracles:*
$\mathsf{San}(.,.,.,.,\mathsf{ssk})$**:** *On input* $(m, \text{MOD}, \sigma, \mathit{pk})$*, it returns* $\mathit{San}(m, \text{MOD}, \sigma, \mathit{pk}, \mathit{ssk})$.
$\mathsf{SaProof}(\mathsf{ssk},.,.,.)$**:** *On input* (m, σ, pk)*, this oracle returns* $\mathit{SaProof}(\mathit{ssk}, m, \sigma, \mathit{pk})$.
$\mathsf{Sa/Si}(b,\mathsf{pk},\mathsf{spk},\mathsf{sk},\mathsf{ssk},.,.,.)$**:** *On input* $(m, \text{ADM}, \text{MOD})$*, if* $\text{ADM}(\text{MOD}) = 0$*, this oracle returns* $\perp$*. Else if* $b = 0$*, this oracle returns* $\mathit{Sig}(\text{MOD}(m), \mathit{sk}, \mathit{spk}, \text{ADM})$*, else if* $b = 1$*, this oracle returns* $\mathit{San}(m, \text{MOD}, \mathit{Sig}(m, \mathit{sk}, \mathit{spk}, \text{ADM}), \mathit{pk}, \mathit{ssk})$.
Let P *be a* SS *of security parameter* k*.* P *is* Trans *secure (or* transparent*) when for any polynomial time adversary* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *wins the following experiment is negligible, where* $\mathit{Sig}(.,\mathit{sk},.,.)$ *and* $\mathit{SiProof}(\mathit{sk},.,.,.)$ *are defined as in Def. 7, and where* $S_{\mathit{Sa/Si}}$ *(resp.* S_{SiProof} *and* S_{SaProof}*) corresponds to the set of all signature outputted by the oracle* $\mathit{Sa/Si}$ *(resp. sending to the oracles* $\mathit{SiProof}$ *and* $\mathit{SaProof}$*):*

$\mathbf{Exp}^{\mathsf{Trans}}_{P,\mathcal{A}}(k)$:
$\mathit{init} \leftarrow \mathit{Init}(1^k)$
$(\mathit{pk}, \mathit{sk}) \leftarrow \mathit{SiGen}(\mathit{init})$
$(\mathit{spk}, \mathit{ssk}) \leftarrow \mathit{SaGen}(\mathit{init})$
$b \xleftarrow{\$} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathit{Sig}(.,\mathit{sk},...),\mathit{San}(.,.,.,.,.,\mathit{ssk}),\mathit{SiProof}(\mathit{sk},.,.,.,.)}(\mathit{pk}, \mathit{spk})$
$\mathit{SaProof}(\mathit{ssk},.,.,.,.),\mathit{Sa/Si}(b,\mathit{pk},\mathit{spk},\mathit{sk},\mathit{ssk},.,.,.,.)$
if $(b = b')$ *and* $(S_{\mathit{Sa/Si}} \cap (S_{\mathit{SiProof}} \cup S_{\mathit{SaProof}}) = \emptyset)$
then return 1, *else return* 0

Unlinkablility: The unlinkablility property ensures that a sanitized signature cannot be linked with the original one. We consider an adversary that has access to the signature oracle, the sanitize oracle, and both the signer and the sanitizer proof oracles. Moreover, the adversary has access to a challenge oracle $\mathsf{LRSan}(b, \mathsf{pk}, \mathsf{ssk}, ., .)$ that depends to a bit b: this oracle takes as input two signatures σ_0 and σ_1, the two corresponding messages m_0 and m_1 and two modification functions MOD_0 and MOD_1 chosen by the adversary. If the two signatures have the same admissible function ADM, if MOD_0 and MOD_1 are admissible according to ADM and if $\text{MOD}_0(m_0) = \text{MOD}_1(m_1)$ then the challenge oracle sanitizes σ_b using MOD_b and returns it. The goal of the adversary is to guess the bit b.

Definition 9 (Unlinkability). *Let the following oracle:*
$\mathsf{LRSan}(b, \mathsf{pk}, \mathsf{ssk}, ., .)$**:** *On input* $((m_0, \text{MOD}_0, \sigma_0)(m_1, \text{MOD}_1, \sigma_1))$*, if for* $i \in \{0,1\}$*,* $\mathit{Ver}(m_i, \sigma_i, \mathit{pk}, \mathit{spk}) = 1$ *and* $\text{ADM}_0 = \text{ADM}_1$ *and* $\text{ADM}_0(\text{MOD}_0) = 1$ *and* $\text{ADM}_1(\text{MOD}_1) = 1$ *and* $\text{MOD}_0(m_0) = \text{MOD}_1(m_1)$*, then this oracle returns* $\mathit{San}(m_b, \text{MOD}_b, \sigma_b, \mathit{pk}, \mathit{ssk})$*, else it returns* 0.
Let P *be a* SS *of security parameter* k*.* P *is* Unlink *secure (or* unlinkable*) when for any polynomial time adversary* $\mathcal{A}$*, the difference between* $1/2$ *and the probability that* $\mathcal{A}$ *wins the following experiment is negligible, where* $\mathit{Sig}(., \mathit{sk}, ., .)$ *and* $\mathit{SiProof}(\mathit{sk}, ., ., .)$ *are defined as in Def. 7 and* $\mathit{San}(., ., ., ., \mathit{ssk})$ *and* $\mathit{SaProof}(\mathit{ssk}, ., ., .)$ *are defined as in Def. 8:*

$\mathbf{Exp}^{\mathsf{Unlink}}_{P,\mathcal{A}}(k)$:
$\mathit{init} \leftarrow \mathit{Init}(1^k)$
$(\mathit{pk}, \mathit{sk}) \leftarrow \mathit{SiGen}(\mathit{init})$
$(\mathit{spk}, \mathit{ssk}) \leftarrow \mathit{SaGen}(\mathit{init})$
$b \xleftarrow{\$} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathit{Sig}(.,\mathit{sk},...),\mathit{San}(.,.,.,.,.,\mathit{ssk})}$
$\mathit{SiProof}(\mathit{sk},.,.,.,.),\mathit{SaProof}(\mathit{ssk},.,.,.,.),\mathit{LRSan}(b,\mathit{pk},\mathit{spk},.,.,.)(\mathit{pk}, \mathit{spk})$
if $(b = b')$ *then return* 1, *else return* 0

Accountability: Standard defintion of accountability is splited into two security experiments: the sanitizer accountability and the signer accountability. In the sanitizer accountability experiment, the adversary has access to the signature oracle and the signer proof oracle. Its goal is to forge a signature such that the signer proof algorithm returns a proof that this signature is not sanitized. To success the experiment, this signature must not come from the signature oracle.

Definition 10 (Sanitizer Accountability). *Let P be a SS of security parameter k. P is $\mathsf{SaAcc\text{-}1}$ secure (or* sanitizer accountable*) when for any polynomial time adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following experiment is negligible, where $\mathsf{Sig}(.,\mathsf{sk},.,.)$ and $\mathsf{SiProof}(\mathsf{sk},.,.,.)$ are defined as in Def. 7, q_{Sig} is the number of calls to the oracle $\mathsf{Sig}(.,\mathsf{sk},.,.)$, $(m_i, \mathrm{ADM}_i, \mathsf{spk}_i)$ is the i^{th} query asked to this oracle and σ_i is the corresponding response:*

$\mathbf{Exp}^{\mathsf{SaAcc\text{-}1}}_{P,\mathcal{A}}(k)$*:*
$\mathsf{init} \leftarrow \mathsf{Init}(1^k)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SiGen}(\mathsf{init})$
$(\mathsf{spk}_*, m_*, \sigma_*) \leftarrow \mathcal{A}^{\mathsf{Sig}(.,\mathsf{sk},.,.),\mathsf{SiProof}(\mathsf{sk},.,.,.)}(\mathsf{pk})$
$\pi^*_{\mathsf{si}} \leftarrow \mathsf{SiProof}(\mathsf{sk}, m_*, \sigma_*, \mathsf{spk}_*)$
if $\forall\, i \in \{1, \ldots, q_{\mathsf{Sig}}\}, (\sigma_* \neq \sigma_i)$
  and $((\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*) = 1))$
  and $(\mathsf{SiJudge}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*, \pi^*_{\mathsf{si}}) \neq 0)$
then return 1*, else return* 0

In the signer accountability experiment, the adversary knows the public key of the sanitizer and has access to the sanitize oracle and the sanitizer proof oracle. Its goal is to forge a signature together with a proof that this signature is sanitized. To success the experiment, this signature must not come from the sanitize oracle.

Definition 11 (Signer Accountability). *Let P be a SS of security parameter k. P is $\mathsf{SiAcc\text{-}1}$ secure (or* signer accountable*) when for any polynomial time adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following experiment is negligible, where $\mathsf{San}(.,.,.,.,\mathsf{ssk})$ and $\mathsf{SaProof}(\mathsf{ssk},.,.,.)$ are defined as in Def. 8 and where q_{San} is the number of calls to the oracle $\mathsf{San}(.,.,.,.,\mathsf{ssk})$, $(m_i, \mathrm{MOD}_i, \sigma_i, \mathsf{pk}_i)$ is the i^{th} query asked to this oracle and σ'_i is the corresponding response:*

$\mathbf{Exp}^{\mathsf{SiAcc\text{-}1}}_{P,\mathcal{A}}(k)$*:*
$\mathsf{init} \leftarrow \mathsf{Init}(1^k)$
$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathsf{SaGen}(\mathsf{init})$
$(\mathsf{pk}_*, m_*, \sigma_*, \pi^*_{\mathsf{si}}) \leftarrow \mathcal{A}^{\mathsf{San}(.,.,.,.,\mathsf{ssk}),\mathsf{SaProof}(\mathsf{ssk},.,.,.)}(\mathsf{spk})$
if $\forall\, i \in \{1, \ldots, q_{\mathsf{San}}\}, (\sigma_* \neq \sigma'_i)$
  and $((\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}) = 1))$
  and $(\mathsf{SiJudge}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}, \pi^*_{\mathsf{si}}) = 0)$
then return 1*, else return* 0

Strong Accountability: Since our definition of sanitizable signature provides a second proof algorithm for the sanitizer, we define two additional security experiments (for signer and sanitizer accountability) to ensure the soundness of the proofs computed by this algorithm. We say that a scheme is strongly accountable when it is signer and sanitizer accountable for both the signer and the sanitizer proof algorithm.

Thus, in our second signer accountability experiment, we consider an adversary that has access to the sanitize oracle and the sanitizer proof oracle. Its goal is to forge a signature such that the sanitizer proof algorithm returns a proof that this signature is sanitized. To win the experiment, this signature must not come from the sanitize oracle.

Definition 12 (Strong Signer Accountability). *Let P be a SS of security parameter k. P is $\mathsf{SiAcc\text{-}2}$ secure when for any polynomial time adversary $\mathcal{A}$, the probability that*

$\mathcal{A}$ wins the following experiment is negligible, where q_{San} is the number of calls to the oracle $\mathsf{San}(.,.,.,.,\mathsf{ssk})$, $(m_i, \mathrm{MOD}_i, \sigma_i, \mathsf{pk}_i)$ is the i^{th} query asked to this oracle and σ'_i is the corresponding response:

$\mathbf{Exp}^{\mathsf{SiAcc\text{-}2}}_{P,\mathcal{A}}(k)$*:*
$\mathsf{init} \leftarrow \mathsf{Init}(1^k)$
$(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathsf{SaGen}(\mathsf{init})$
$(\mathsf{pk}_*, m_*, \sigma_*) \leftarrow \mathcal{A}^{\mathsf{San}(.,.,.,.,\mathsf{ssk}),\mathsf{SaProof}(\mathsf{ssk},.,.,.)}(\mathsf{spk})$
$\pi^*_{\mathsf{sa}} \leftarrow \mathsf{SaProof}(\mathsf{ssk}, m_*, \sigma_*, \mathsf{pk}_*)$
if $\forall\, i \in \{1, \ldots, q_{\mathsf{San}}\}, (\sigma_* \neq \sigma'_i)$
  and $((\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}) = 1))$
  and $(\mathsf{SaJudge}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}, \pi^*_{\mathsf{sa}}) \neq 1)$
then return 1*, else return* 0

P *is* strong signer accountable *when it is both* $\mathsf{SiAcc\text{-}1}$ *and* $\mathsf{SiAcc\text{-}2}$ *secure.*

Finally, in our second sanitizer accountability experiment, we consider an adversary that knows the public key of the signer and has access to the signer oracle and the signer proof oracle. Its goal is to sanitize a signature with a proof that this signature is not sanitized. To win the experiment, this signature must not come from the signer oracle.

Definition 13 (Strong Sanitizer Accountability). *Let P be a SS of security parameter k. P is $\mathsf{SaAcc\text{-}2}$ secure when for any polynomial time adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following experiment is negligible, $\mathsf{Sig}(.,\mathsf{sk},.,.)$ and $\mathsf{SiProof}(\mathsf{sk},.,.,.)$ are defined as in Def. 7, q_{Sig} is the number of calls to the oracle $\mathsf{Sig}(.,\mathsf{sk},.,.)$, $(m_i, \mathrm{ADM}_i, \mathsf{spk}_i)$ is the i^{th} query asked to this oracle and σ_i is the corresponding response:*

$\mathbf{Exp}^{\mathsf{SaAcc\text{-}2}}_{P,\mathcal{A}}(k)$*:*
$\mathsf{init} \leftarrow \mathsf{Init}(1^k)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SaGen}(\mathsf{init})$
$(\mathsf{spk}_*, m_*, \sigma_*, \pi^*_{\mathsf{sa}}) \leftarrow \mathcal{A}^{\mathsf{Sig}(.,\mathsf{sk},.,.),\mathsf{SiProof}(\mathsf{sk},.,.,.)}(\mathsf{spk})$
if $\forall\, i \in \{1, \ldots, q_{\mathsf{Sig}}\}, (\sigma_* \neq \sigma_i)$
  and $((\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*) = 1))$
  and $(\mathsf{SaJudge}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*, \pi^*_{\mathsf{sa}}) = 1)$
then return 1*, else return* 0

P *is* strong sanitizer accountable *when it is both* $\mathsf{SaAcc\text{-}1}$ *and* $\mathsf{SaAcc\text{-}2}$ *secure.*

3 Schemes

3.1 An Efficient Verifiable Ring Signature: EVeR

We present our VRS scheme called EVeR (for *Efficient VErifiable Ring signature*). It is based on the DDH assumption and uses a NIZKP of equality of two discrete logarithms out of n elements. We show how to build this NIZKP: let G be a group of prime order p, n be an integer and the following language be:
$\mathcal{L}_n = \left\{\{(h_i, z_i, g_i, y_i)\}_{1 \leq i \leq n} : \exists 1 \leq i \leq n, (h_i, z_i, g_i, y_i) \in G^4; \log_{g_i}(y_i) = \log_{h_i}(z_i)\right\}$

Consider the case $n = 1$. In [13], authors present an interactive zero-knowledge proof of knowledge system for the language $\mathcal{L}_1$. It proves the equality of two discrete logarithms. For example using $(h, z, g, y) \in \mathcal{L}_1$, a prover convinces a verifier

that $\log_g(y) = \log_h(z)$. The witness used by the prover is $x = \log_g(y)$. This proof system is a *sigma protocol* in the sense that there are only three interactions: the prover sends a commitment, the verifier sends a challenge, and the prover returns a response.

To transform the proof system of $\mathcal{L}_1$ into a generic proof system of any $\mathcal{L}_n$, we use the generic transformation given in [14]. For any language $\mathcal{L}$ and any integer n, the authors show how to transform a proof that an element is in $\mathcal{L}$ into a proof that one out of n element is in $\mathcal{L}$ under the condition that the proof is a sigma protocol. Note that the resulting proof system is also a sigma protocol.

The final step is to transform it into a non-interactive proof system. We use the well-known Fiat-Shamir transformation [15]. This transformation outputs a non interactive proof system from any interactive proof system that is a sigma protocol. The resulting proof system is complete, sound and zero-knowledge in the random oracle model. Finally, we obtain the following scheme.

Scheme 1 (LogEq_n) *Let G be a group of prime order p, $H : \{0,1\} \to \mathbb{Z}_p^*$ be a hash function and n be an integer. We define the NIZKP system $\mathsf{LogEq}_n = (\mathsf{LEprove}_n, \mathsf{LEverif}_n)$ for $\mathcal{L}_n$ by:*

$\mathsf{LEprove}_n(\{(h_i, z_i, g_i, y_i\}_{1\leq i\leq n}, x)$. *We denote by j the integer such that $x = \log_{g_j}(y_j) = \log_{h_j}(z_j)$. This algorithm picks $r_j \xleftarrow{\$} \mathbb{Z}_p^*$, computes $R_j = g_j^{r_j}$ and $S_j = h_j^{r_j}$. For all $i \in \{1,\dots,n\}$ and $i \neq j$, it picks $c_i \xleftarrow{\$} \mathbb{Z}_p^*$ and $\gamma_i \xleftarrow{\$} \mathbb{Z}_p^*$, and computes $R_i = g_i^{\gamma_i}/y_i^{c_i}$ and $S_i = h_i^{\gamma_i}/z_i^{c_i}$. It computes $c = H(R_1||S_1||\dots||R_n||S_n)$. It then computes $c_j = c/(\prod_{i=1;i\neq j}^{n} c_i)$ and $\gamma_j = r_j + c_j \cdot x$. It outputs $\pi = (\{R_i, S_i, c_i, \gamma_i\}_{1\leq i\leq n})$.*

$\mathsf{LEverif}_n(\{(h_i, z_i, g_i, y_i\}_{1\leq i\leq n}, \pi)$. *Using $\pi = (\{R_i, S_i, c_i, \gamma_i\}_{1\leq i\leq n})$. If $\prod_{i=1;i\neq j}^{n} c_i \neq H(R_1||S_1||\dots||R_n||S_n)$ then it returns 0. Else if there exists $i \in \{1,\dots,n\}$ such that $g_i^{\gamma_i} \neq R_i \cdot y_i^{c_i}$ or $h_i^{\gamma_i} \neq S_i \cdot z_i^{c_i}$ then it returns 0. Else it returns 1.*

Theorem 1. *The NIZKP LogEq_n is a proof of knowledge, moreover it is complete, sound, and zero-knowledge in the random oracle model.*

The proof of this theorem is a direct implication of [13], [14] and [15].

Using this proof system, we build our VRS scheme called EVeR:

Scheme 2 (Efficient Verifiable Ring Signature (EVeR)) EVeR *is a VRS defined by:*

$\mathsf{V.Init}(1^k)$: *It generates a prime order group setup (G, p, g) and a hash function $H : \{0,1\}^* \to G$. It returns a setup value $\mathsf{init} = (G, p, g, H)$.*

$\mathsf{V.Gen}(\mathsf{init})$: *It picks $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$, computes $\mathsf{pk} = g^{\mathsf{sk}}$ and returns a pair of signer public/private keys $(\mathsf{pk}, \mathsf{sk})$.*

$\mathsf{V.Sig}(L, m, \mathsf{sk})$: *It picks $r \xleftarrow{\$} \mathbb{Z}_p^*$, it computes $h = H(m||r)$ and $z = h^{\mathsf{sk}}$, it runs $P \leftarrow \mathsf{LEprove}_{|L|}(\{(h, z, g, \mathsf{pk}_l)\}_{\mathsf{pk}_l\in L}, \mathsf{sk})$ and returns $\sigma = (r, z, P)$.*

$\mathsf{V.Ver}(L, m, \sigma)$: *It parses $\sigma = (r, z, P)$, computes $h = H(m||r)$ and returns $b \leftarrow \mathsf{LEverif}_{|L|}(\{(h, z, g, \mathsf{pk}_l)\}_{\mathsf{pk}_l\in L}, P)$.*

$\mathsf{V.Proof}(L, m, \sigma, \mathsf{pk}, \mathsf{sk})$: *It parses $\sigma = (r, z, P)$, computes $h = H(m||r)$ and $\bar{z} = h^{\mathsf{sk}}$, runs $\bar{P} \leftarrow \mathsf{LEprove}_1(\{(h, \bar{z}, g, \mathsf{pk})\}, \mathsf{sk})$ and returns $\pi = (\bar{z}, \bar{P})$.*

$\mathsf{V.Judge}(L, m, \sigma, \mathsf{pk}, \pi)$: *It parses* $\sigma = (r, z, P)$ *and* $\pi = (\bar{z}, \bar{P})$*, computes* $h = H(m||r)$ *and runs* $b \leftarrow \mathsf{LEprove}_1(\{(h, \bar{z}, g, \mathsf{pk})\}, \pi)$*. If* $b \neq 1$ *then it returns* $\perp$*. Else, if* $z = \bar{z}$ *then it returns* 1*, else it returns* 0*.*

All users have an ElGamal key pair $(\mathsf{pk}, \mathsf{sk})$ such that $\mathsf{pk} = g^{\mathsf{sk}}$ where g is a generator of a prime order group. To sign a message m according to a set of public key L using her key pair $(\mathsf{pk}, \mathsf{sk})$, Alice chooses a random r and computes $h = H(m||r)$ and $z = h^{\mathsf{sk}}$ where H is an universal hash function. Alice produces a proof π that there exists $\mathsf{pk}_l \in L$ such that $\log_g(\mathsf{pk}_l) = \log_h(z)$ using the NIZKP $\mathsf{LogEq}_{|L|}$ where $|L|$ denotes the cardinal of L . The signature is the triplet (r, z, π). To verify a signature, it suffices to verify the proof π according to L, m and the other parts of the signature. To prove that she is the signer of the message m, Alice generates a proof that $\log_g(\mathsf{pk}) = \log_h(z)$ using the NIZKP LogEq_1. Verifying this proof, a judge is convinced that $z = h^{\mathsf{sk}}$. We then consider a second signature (r', z', π') of a message m' produced from another key pair $(\mathsf{pk}', \mathsf{sk}')$. We set $h' = H(m'||r')$, and we recall that $z' = (h')^{\mathsf{sk}'}$. To prove that she is not the signer of m', Alice computes $\bar{z}' = (h')^{\mathsf{sk}}$ and she generates a proof that $\log_g(\mathsf{pk}) = \log_{h'}(\bar{z}')$. Since $\bar{z}' \neq z'$, Alice proves that $\log_g(\mathsf{pk}) \neq \log_{h'}(z')$, then she is not the signer of (r', z', π').

Theorem 2. EVeR *is unforgeable, anonymous, accountable and non-usurpable under the DDH assumption in the random oracle model.*

We give the intuition of the security properties, the proof of the theorem is given Appendix C:

Unforgeability: The scheme is unforgeablie since nobody can prove that $\log_g(\mathsf{pk}_l) = \log_h(z)$ without the knowledge of $\mathsf{sk} = \log_h(z)$.

Anonymity: Breaking the anonymity of such a signature is equivalent to break the DDH assumption. Indeed, to link a signature $z = h^{\mathsf{sk}}$ with the corresponding public key of Alice $\mathsf{pk} = g^{\mathsf{sk}}$, an attacker must solve the DDH problem on the instance (pk, h, z). Moreover, note that since the value r *randomizes* the signature, it is not possible to link two signatures of the same message produced by Alice.

Accountability: To break the accountability, an adversary must to forge a valid signature (*i.e.* to prove that there exists pk_l in the group such that $\log_g(\mathsf{pk}_l) \neq \log_h(z)$) and to prove that he is not the signer (*i.e.* $\log_g(\mathsf{pk}) \neq \log_h(z)$ where pk is the public key chosen by the adversary). However, since the adversary does not know the secret keys of the other members of the group, it would have to break the soundness of LogEq to win the experiment, which is not possible.

Non-usurpability: (-non-usu-1) no adversary is able forge a proof that he is the signer of a signature produced by another user since it is equivalent to prove a false statement using a sound NIZKP. (-non-usu-2) the proof algorithm run by a honest user with the public key pk returns a proof that this user is the signer of a given signature only if $\log_g(\mathsf{pk}) = \log_h(z)$. Since no adversary is able to compute z such that $\log_g(\mathsf{pk}) = \log_h(z)$ without the corresponding secret key, no adversary is able to break the non-usuprability of EvER.

3.2 Our Unlinkable Sanitizable Signature Scheme: GUSS

We present our USS instantiated by a digital signature (DS) scheme and a VRS.

Scheme 3 (Generic Unlinkable Sanitizable Signature (GUSS)) *Let D be a deterministic digital signature scheme and V be a verifiable ring signature scheme such that:* $D = (\mathit{D.Init}, \mathit{D.Gen}, \mathit{D.Sig}, \mathit{D.Ver})$ $V = (\mathit{V.Init}, \mathit{V.Gen}, \mathit{V.Sig}, \mathit{V.Ver}, \mathit{V.Proof}, \mathit{V.Judge})$ GUSS *instantiated with (D, V) is a sanitizable signature scheme defined by:*

$\mathsf{Init}(1^k)$: *It runs $\mathit{init}_d \leftarrow \mathit{D.Init}(1^k)$ and $\mathit{init}_v \leftarrow \mathit{V.Init}(1^k)$, it returns $\mathit{init} = (\mathit{init}_d, \mathit{init}_v)$.*

$\mathsf{SiGen}(\mathsf{init})$: *It parses $\mathit{init} = (\mathit{init}_d, \mathit{init}_v)$, runs $(\mathit{pk}_d, \mathit{sk}_d) \leftarrow \mathit{SiGen}(\mathit{init}_d)$ and $(\mathit{pk}_v, \mathit{sk}_v) \leftarrow \mathit{V.Gen}(\mathit{init}_v)$ returns $(\mathit{pk}, \mathit{sk})$ where $\mathit{pk} = (\mathit{pk}_d, \mathit{pk}_v)$ and $\mathit{sk} = (\mathit{sk}_d, \mathit{sk}_v)$.*

$\mathsf{SaGen}(\mathsf{init})$: *It parses $\mathit{init} = (\mathit{init}_d, \mathit{init}_v)$ and runs $(\mathit{spk}, \mathit{ssk}) \leftarrow \mathit{V.Gen}(\mathit{init}_v)$. It returns $(\mathit{spk}, \mathit{ssk})$.*

$\mathsf{Sig}(m, \mathsf{sk}, \mathsf{spk}, \textsc{Adm})$: *It parses $\mathit{sk} = (\mathit{sk}_d, \mathit{sk}_v)$. It first computes the fixed message part $M \leftarrow \textsc{Fix}_{\textsc{Adm}}(m)$ and runs $\sigma_1 \leftarrow \mathit{D.Sig}(\mathit{sk}_d, (M||\textsc{Adm}||\mathit{pk}||\mathit{spk}))$ and $\sigma_2 \leftarrow \mathit{V.Sig}(\{\mathit{pk}_v, \mathit{spk}\}, \mathit{sk}_v, (\sigma_1||m))$. It returns $\sigma = (\sigma_1, \sigma_2, \textsc{Adm})$.*

$\mathsf{San}(m, \textsc{Mod}, \sigma, \mathsf{pk}, \mathsf{ssk})$: *It parses $\sigma = (\sigma_1, \sigma_2, \textsc{Adm})$ and $\mathit{pk} = (\mathit{pk}_d, \mathit{pk}_v)$. This algorithm first computes the modified message $m' \leftarrow \textsc{Mod}(m)$ and it runs $\sigma_2' \leftarrow \mathit{V.Sig}(\{\mathit{pk}_v, \mathit{spk}\}, \mathit{ssk}, (\sigma_1||m'))$. It returns $\sigma' = (\sigma_1, \sigma_2', \textsc{Adm})$.*

$\mathsf{Ver}(m, \sigma, \mathsf{pk}, \mathsf{spk})$: *It parses $\sigma = (\sigma_1, \sigma_2, \textsc{Adm})$ and it computes the fixed message part $M \leftarrow \textsc{Fix}_{\textsc{Adm}}(m)$. It then runs $b_1 \leftarrow \mathit{D.Ver}(\mathit{pk}_d, (M||\textsc{Adm}||\mathit{pk}||\mathit{spk}), \sigma_1)$ and $b_2 \leftarrow \mathit{V.Ver}(\{\mathit{pk}_d, \mathit{spk}\}, (\sigma_1||m), \sigma_2)$. It returns $b = (b_1 \wedge b_2)$.*

$\mathsf{SiProof}(\mathsf{sk}, m, \sigma, \mathsf{spk})$: *It parses $\sigma = (\sigma_1, \sigma_2, \textsc{Adm})$ and the key $\mathit{sk} = (\mathit{sk}_d, \mathit{sk}_v)$. It runs $\pi_{\mathit{si}} \leftarrow \mathit{V.Proof}(\{\mathit{pk}_v, \mathit{spk}\}, (m||\sigma_1), \sigma_2, \mathit{pk}_v, \mathit{sk}_v)$ and returns it.*

$\mathsf{SaProof}(\mathsf{ssk}, m, \sigma, \mathsf{pk})$: *It parses the signature $\sigma = (\sigma_1, \sigma_2, \textsc{Adm})$. It runs $\pi_{\mathit{sa}} \leftarrow \mathit{V.Proof}(\{\mathit{pk}_v, \mathit{spk}\}, (m||\sigma_1), \sigma_2, \mathit{spk}, \mathit{ssk})$ and returns it.*

$\mathsf{SiJudge}(m, \sigma, \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{si}})$: *It parses $\sigma = (\sigma_1, \sigma_2, \textsc{Adm})$ and $\mathit{pk} = (\mathit{pk}_d, \mathit{pk}_v)$. It runs $b \leftarrow \mathit{V.Judge}(\{\mathit{pk}_v, \mathit{spk}\}, (m||\sigma_1), \sigma_2, \mathit{pk}_v, \pi_{\mathit{si}})$ and returns it.*

$\mathsf{SaJudge}(m, \sigma, \mathsf{pk}, \mathsf{spk}, \pi_{\mathsf{sa}})$: *It parses $\sigma = (\sigma_1, \sigma_2, \textsc{Adm})$ and $\mathit{pk} = (\mathit{pk}_d, \mathit{pk}_v)$. It runs $b \leftarrow \mathit{V.Judge}(\{\mathit{pk}_v, \mathit{spk}\}, (m||\sigma_1), \sigma_2, \mathit{spk}, \pi_{\mathit{sa}})$ and returns $(1 - b)$.*

The signer secret key $\mathsf{sk} = (\mathsf{sk}_d, \mathsf{sk}_v)$ contains a secret key sk_d compatible with the DS scheme and a secret key sk_v compatible the VRS scheme. The signer public key $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$ contains the two corresponding public keys. The sanitizer public/secret key pair $(\mathsf{spk}, \mathsf{ssk})$ is generated as in the VRS scheme.

Let m be a message and M be the *fixed part* chosen by the signer according to the admissible function $\textsc{Adm}$. To sign m, the signer first signs M together with the public key of the sanitizer spk and the admissible function $\textsc{Adm}$ using the DS scheme. We denote this signature by σ_1. The signer then signs in σ_2 the full message m together with σ_1 using the VRS scheme for the set of public keys $L = \{\mathsf{pk}_v, \mathsf{spk}\}$. On the other words, he anonymously signs $(\sigma_1||m)$ within a group of two users: the signer and the sanitizer. The final sanitizable signature is $\sigma = (\sigma_1, \sigma_2)$. The verification algorithm is in two steps: it verifies the signature σ_1 and it verifies the anonymous signature σ_2.

To sanitize this signature $\sigma = (\sigma_1, \sigma_2)$, the sanitizer chooses an admissible message m' according to $\textsc{Adm}$ (*i.e.* m and m' have the same fixed part). He then anonymously signs m' together with σ_1 using the VRS for the group $L = \{\mathsf{pk}_v, \mathsf{spk}\}$ using the secret key ssk. We denote by σ_2' this signature. The final sanitized signature is $\sigma' = (\sigma_1, \sigma_2')$.

Theorem 3. *For any deterministic and unforgeable DS scheme D and any unforgeable, anonymous, accountable and non-usurpable VRS scheme V,* GUSS *instantiated with (D, V) is immutable, transparent, strongly accountable and unlinkalbe.*

We give the intuition of the security properties, the proof of the theorem is given in Appendix D:

Transparency: According to the anonymity of σ_2 and σ'_2, nobody can guess if a signature come from the signer or the sanitizer, and since both signatures have the same structure, we cannot guess whether a signature is sanitized or not.

Immutability: Since it is produced by an unforgeable DS scheme, nobody can forge the signature σ_1 of the fixed part M without the signer secret key. Thus the sanitizer cannot change the fixed part of the signatures. Moreover, since σ_1 signs the public key of the sanitizer in addition to M, the other users can not forge a signature of an admissible message using σ_1.

Unlinkability: An adversary knows (i) two signatures σ^0 and σ^1 that have the same fixed part M according to the same function ADM for the same sanitizer and (ii) the sanitized signature $\sigma' = (\sigma'_1, \sigma'_2)$ computed from σ^b for a given admissible message m' and an unknown bit b. To achieve unlinkability, it must be hard to guess b. Since the DS scheme is deterministic, the two signatures $\sigma^0 = (\sigma^0_1, \sigma^0_2)$ and $\sigma^1 = (\sigma^1_1, \sigma^1_2)$ have the same first part (*i.e.* $\sigma^0_1 = \sigma^1_1$). As it was shown before, the σ' has the same first part σ'_1 as the original one, thus $\sigma'_1 = \sigma^0_1 = \sigma^1_1$ and σ'_1 leaks no information about b. On the other hand, the second part of the sanitized signature σ'_2 is computed from the modified message m' and the first part of the original signature. Since $\sigma^0_1 = \sigma^1_1$, we deduce that σ'_2 leaks no information about b. Finally, the best strategie of the adversary is to randomely guess b.

(Strong) Accountability: the signer must be able to prove the provenance of a signature. It is equivalent to break the anonymity of the second parts σ_2 of this signature: if it was created by the signer then it is the original signature, else it was created by the sanitizer and it is a sanitized signature. By definition, the VRS scheme used to generate σ_2 provides a way to prove whether a user is the author of a signature or not. GUSS uses it in its proof algorithm to achieve accountability. Note that since the sanitizer uses the same VRS scheme to sanitize a signature, it also can prove the origin of a given signature to achieve the strong accountability.

4 Conclusion

In this paper, we revisit the notion of verifiable ring signatures. We improve its properties of verifiability, we give a security model for this primitive and we design a simple, efficient and secure scheme named EvER. We extend the security model of sanitizable signatures in order to allow the sanitizer to prove the origin of a signature. Finally, we design a generic unlinkable sanitizable signature scheme named GUSS based on verifiable ring signatures. This scheme is twice as efficient as the best scheme of the literature. In the future, we aim at find other applications to verifiable ring signatures that are secure in our model.

References

1. Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. *Sanitizable Signatures*, pages 159–177. Springer Berlin Heidelberg, 2005.
2. Dan Boneh. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *LNCS*. Springer, 1998. Invited paper.
3. Christina Brzuska, Heike Busch, Oezguer Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, and Dominique Schröder. *Redactable Signatures for Tree-Structured Data: Definitions and Constructions*. Springer Berlin Heidelberg, 2010.
4. Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 317–336. Springer, March 2009.
5. Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. *Security of Sanitizable Signatures Revisited*, pages 317–336. Springer Berlin Heidelberg, 2009.
6. Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 444–461. Springer, May 2010.
7. Christina Brzuska, Henrich C. Pöhls, and Kai Samelin. *Non-interactive Public Accountability for Sanitizable Signatures*, pages 178–193. Berlin, Heidelberg, 2013.
8. Christina Brzuska, Henrich C. Pöhls, and Kai Samelin. *Efficient and Perfectly Unlinkable Sanitizable Signatures without Group Signatures*, pages 12–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.
9. Sébastien Canard and Amandine Jambert. *On Extended Sanitizable Signature Schemes*, pages 179–194. Springer Berlin Heidelberg, 2010.
10. Sébastien Canard, Amandine Jambert, and Roch Lescuyer. *Sanitizable Signatures with Several Signers and Sanitizers*, pages 35–52. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
11. Sbastien Canard, Berry Schoenmakers, Martijn Stam, and Jacques Traor. List signature schemes. *Discrete Applied Mathematics*, 154(2):189 – 201, 2006.
12. Z. Changlun, L. Yun, and H. Dequan. A new verifiable ring signature scheme based on nyberg-rueppel scheme. In *2006 8th international Conference on Signal Processing*, volume 4, 2006.
13. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, August 1993.
14. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO94*, volume 839 of *LNCS*. Springer, 1994.
15. Amos Fiat and Adi Shamir. *Advances in Cryptology — CRYPTO' 86: Proceedings*, chapter How To Prove Yourself: Practical Solutions to Identification and Signature Problems, pages 186–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987.
16. N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schrder, and M. Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In *Public-Key Cryptography PKC 2016*, LNCS. Springer, 2016.
17. Georg Fuchsbauer and David Pointcheval. *Anonymous Proxy Signatures*. Springer Berlin Heidelberg, 2008.
18. Robert Johnson, David Molnar, Dawn Song, and David Wagner. *Homomorphic Signature Schemes*, pages 244–262. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

19. Russell W. F. Lai, Tao Zhang, Sherman S. M. Chow, and Dominique Schröder. *Efficient Sanitizable Signatures Without Random Oracles*, pages 363–380. Springer International Publishing, 2016.
20. K. C. Lee, H. A. Wen, and T. Hwang. Convertible ring signature. *IEE Proceedings - Communications*, 152(4):411–414, 2005.
21. Jiqiang Lv and Xinmei Wang. *Verifiable Ring Signature*, pages 663–665. DMS Proceedings, 2003.
22. Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, December 2001.
23. Ron Steinfeld, Laurence Bull, and Yuliang Zheng. *Content Extraction Signatures*, pages 285–304. Springer Berlin Heidelberg, 2002.
24. Shangping Wang, Rui Ma, Yaling Zhang, and Xiaofeng Wang. Ring signature scheme based on multivariate public key cryptosystems. *Computers and Mathematics with Applications*, 62(10):3973 – 3979, 2011.

A Cryptographic Backgroond

Definition 14 (DDH [2]). *Let $\mathbb{G}$ be a multiplicative group of prime order p and $g \in \mathbb{G}$ be a generator. Given an instance $h = (g^a, g^b, g^z)$ for unknown $a, b, z \xleftarrow{\$} \mathbb{Z}_p^*$, the* Decisional Diffie-Hellman *(DDH)* problem *is to decide whether $z = a \cdot b$ or not. The DDH* hypothesis *states that there exists no PPT algorithm that solves DDH problem.*

We recall the notion of deterministic digital signature, and we recall the deterministic version of the Schnorr's signature.

Definition 15 ((Deterministic) Digital Signature (DS)). *A* Digital Signature *scheme S is a tuple of 4 algorithms defined as follows:*

D.Init(1^k): *It returns a setup value init.*
SiGen(init): *It returns a pair of signer public/private keys (pk, sk).*
D.Sig(m, sk): *This algorithm computes a signature σ of m using the key sk.*
D.Ver(pk, m, σ): *It returns a bit b: if the signature σ of m is valid according to pk then $b = 1$, else $b = 0$.*

Such a scheme is unforgeable *when no polynomial adversary wins the following experiment with non-negligible probability where D.Sig$(\cdot, sk)$ is a signature oracle, q_S is the number of queries to this oracle and σ_i is the i^{th} signature computed by this oracle:*

$\mathbf{Exp}^{\mathsf{unf}}_{S,\mathcal{A}}(k)$*:*
$init \leftarrow D.Init(1^k)$
$(pk, sk) \leftarrow SiGen(init)$
$(m_*, \sigma_*) \leftarrow \mathcal{A}^{D.Sig(\cdot, sk)}(pk)$
if $(D.Ver(\sigma_*, m_*, =)1)$ *and* $(\forall\, i \in \{1, \ldots, q_S\}, \sigma_i \neq \sigma_*)$
then return 1*, else return* 0

Moreover, such a scheme is deterministic when the algorithm D.Sig(m, sk) is deterministic. As it is mentioned in [16], any DS scheme can be transformed into a deterministic DS scheme without lost of efficiency or security using a pseudo random function, that can be simulated by a hash function in the random oracle model.

Definition 16 (Deterministic Schnorr's Signature). *The (Deterministic) Schnorr's Signature is defined by the following algorithms:*

$\mathsf{D.Init}(1^k)$: *It returns a setup value* $\mathsf{init} = (G, p, g, H)$ *where* G *is a group of prime order* p*,* $g \in G$ *and* $H : \{0,1\} \to \mathbb{Z}_p^*$ *is a hash function.*

$\mathsf{SiGen}(\mathsf{init})$: *It picks* $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$*, computes* $\mathsf{pk} = g^{\mathsf{sk}}$ *and returns* $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{D.Sig}(m, \mathsf{sk})$: *It compute the* $r = H(m||\mathsf{sk})$*,* $R = g^r$*,* $z = r + \mathsf{sk} \cdot H(R||m)$ *and returns* $\sigma = (R, z)$.

$\mathsf{D.Ver}(\mathsf{pk}, m, \sigma)$: *It parse* $\sigma = (R, z)$*, if* $g^z = R \cdot \mathsf{pk}^{H(R||m)}$ *then it returns* 1*, else* 0.

This DS scheme is deterministic and unforgeable under the DDH assumption in the random oracle model.

A zero-knowledge proof (ZKP) allows a prover knowing a witness to convince a verifier that a statement s is in a given language without leak any information except s. Such a proof is a proof of knowledge (PoK) when the verifier is also convinced that the prover knows the witness w. We recall the definition of a non-interactive zero-knowledge proof of knowledge.

Definition 17 (NIZKP). *A* non-interactive ZKP *(NIZKP) For a language* $\mathcal{L}$ *is a couple of algorithms* $(ZKPpro, ZKPver)$ *such that:*

$\mathsf{Prove}(s, w)$**.** *This algorithm outputs a proof* π *that* $s \in \mathcal{L}$ *using the witness* w.

$\mathsf{Verify}(s, \pi)$**.** *This algorithm checks whether* π *is a valid proof that* $s \in \mathcal{L}$ *and outputs a bit.*

A NIZKP proof verifies the following properties:

Completeness *For any statement* $s \in \mathcal{L}$ *and the corresponding witness* w*, we have that* $\mathsf{Verify}(s, \mathsf{Prove}(s, w)) = 1$.

Soundness *There is no polynomial time adversary* $\mathcal{A}$ *such that* $\mathcal{A}(\mathcal{L})$ *outputs* (s, π) *such that* $\mathsf{Verify}(s, \pi) = 1$ *and* $x \notin \mathcal{L}$ *with non-negligible probability.*

Zero-knowledge. *A proof* π leaks no information*, i.e. there exists a PPT algorithm* Sim *(called the* simulator*) such that outputs of* $\mathsf{Prove}(s, w)$ *and the outputs of* $\mathsf{Sim}(s)$ *follow the same probability distribution.*

Moreover, such a proof is a proof of knowledge *when for any* $s \in \mathcal{L}$ *and the corresponding witness* w*, any bit-string input* $\text{in} \in \{0,1\}^*$ *and any algorithm* $\mathcal{A}(s, \text{in})$ *there exists a knowledge extractor* $\mathcal{E}$ *such that the probability that* $\mathcal{E}^{\mathcal{A}(s,\text{in})}(s)$ *outputs the witness* w *given access to oracle* $\mathcal{A}(s, \text{in})$ *is as high as the probability that* $\mathcal{A}(s, \text{in})$ *outputs a proof* π *such that* $\mathsf{Verify}(s, \pi) = 1$.

B First experiment for non-usurpability

Definition 18 (n-non-usu-1 experiment). *Let* P *be a* SS *of security parameter* k*.* P *is* n*-non-usu-1 secure when for any polynomial time adversary* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *wins the following experiment is negligible, where* $\mathsf{V.Sig}(\cdot,\cdot,\cdot)$ *and* $\mathsf{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$ *and where* q_S *is the number of calls to the oracle* $\mathsf{V.Sig}(\cdot,\cdot,\cdot)$ *and* (L_i, l_i, m_i) *(resp.* σ_i*) is is the* i^{th} *query to this oracle (resp. signature outputted by this oracle):*

$\mathbf{Exp}^{n\text{-non-usu-1}}_{P,\mathcal{A}}(k)$:
$\mathit{init} \leftarrow \mathit{V.Init}(1^k)$
$\forall 1 \leq i \leq n, (\mathit{pk}_i, \mathit{sk}_i) \leftarrow \mathit{V.Gen}(\mathit{init})$
$(L_*, m_*, \sigma_*, l_*, \pi_*) \leftarrow \mathcal{A}^{\mathit{V.Sig}(\cdot,\cdot,\cdot),\mathit{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)}(\{\mathit{pk}_i\}_{1\leq i\leq n})$
$\pi \leftarrow \mathit{V.Proof}(L_*, m_*, \sigma_*, \mathit{pk}, \mathit{sk})$
if $(\mathit{V.Ver}(L_*, \sigma_*, m_*) = 1)$ *and*
$(\mathit{V.Judge}(L_*, m_*, \sigma_*, \mathit{pk}_{l_*}, \pi_*) = 1)$ *and*
$(\forall\, i \in \{1, \ldots, q_S\}, (L_i, l_i, m_i, \sigma_i) = (L_*, l_*, m_*, \sigma_*))$
then return 1*, else return* 0

C Security proofs of EVeR

Lemma 1. EVeR *is* n*-unf secure for any polynomially bounded* n *under the DL assumption in the random oracle model.*

Proof. We recall that since LogEq_n is valid, then for any $s \in \mathcal{L}$ and the corresponding witness w, for any bit-string input $\mathrm{in} \in \{0,1\}^*$ and any algorithm $\mathcal{A}(\mathrm{in})$ there exists a knowledge extractor $\mathcal{E}$ such that the probability that $\mathcal{E}^{\mathcal{A}(\mathrm{in})}(k)$ outputs the witness w given access to oracle $\mathcal{A}(\mathrm{in})$ is as high as the probability that $\mathcal{A}(\mathrm{in})$ outputs a proof π such that $\mathsf{Verify}(s, \pi) = 1$. Moreover, since LogEq_n is zero-knowledge there exists a PPT algorithm Sim (called the *simulator*) such that outputs of $\mathsf{Prove}(s, w)$ and the outputs of $\mathsf{Sim}(s)$ follow the same probability distribution.

Suppose that there exists an adversary $\mathcal{A} \in \mathrm{POLY}(k)$ such the advantage $\lambda(k) = \mathsf{Pr}[\mathsf{Exp}^{n\text{-unf}}_{\mathrm{EVeR},\mathcal{A}}(k) = 1]$ is non-negligible. We show how to build an algorithm $\mathcal{B} \in \mathrm{POLY}(k)$ that solve the DL problem with non-negligible probability.

$\mathcal{B}$ *construction:* $\mathcal{B}$ receives the input (G, p, g, y) where g is the generator of the group G of prime order p and y is an element of G. For all $i \in \{1, \cdots, n\}$, it picks $x_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\mathsf{pk}_i = y^{x_i}$. $\mathcal{B}$ initializes an empty list H_{list}. $\mathcal{B}$ runs $x' \leftarrow \mathcal{E}^{\mathcal{A}'(\{\mathsf{pk}_i\}_{1\leq i\leq n})}(k)$ where $\mathcal{A}'$ is the following algorithm:

Algorithm $\mathcal{A}'(\{\mathsf{pk}_i\}_{1\leq i\leq n})$**:** It runs $(L_*, \sigma_*, m_*) \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{1\leq i\leq n})$. It simulates the oracles to $\mathcal{A}$ as follows:

- **Random oracle** $H(.)$**:** On the i^{th} input M_i, if $\exists j < i$ such that $M_j = M_i$ then it sets $u_j = u_i$. Else it picks $u_i \xleftarrow{\$} \mathbb{Z}_p^*$. Finally, it returns g^{u_i}.
- **Oracle** $\mathbf{V.Sig}(\cdot,\cdot,\cdot)$**:** On the i^{th} input (L_i, l_i, m_i), it picks $r_i \xleftarrow{\$} \mathbb{Z}_p^*$. It computes $h_i = H(m_i||r_i)$ using the oracle $H(.)$, then there exists j such that $m'_i||r'_i = M_j$. It computes $z_i = \mathsf{pk}_{l_i}^{u_j}$ and it runs $P_i \leftarrow \mathsf{Sim}(\{(h_i, z_i, g, \mathsf{pk}_l)\}_{\mathsf{pk}_l \in L_i})$. It returns (r_i, z_i, P_i) to $\mathcal{A}$.
- **Oracle** $\mathbf{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$**:** On the i^{th} input $(L'_i, m'_i, \sigma'_i, l'_i)$, it parses $\sigma'_i = (r'_i, z'_i, P'_i)$. It computes $h'_i = H(m'_i||r'_i)$ using the oracle $H(.)$, then there exists j such that $m'_i||r'_i = M_j$ It computes $\bar{z}_i = \mathsf{pk}_{l'_i}^{u_j}$ and it runs $P'_i \leftarrow \mathsf{Sim}((h'_i, \bar{z}_i, g, \mathsf{pk}_{l'_i})$. It returns $(\bar{z}_i, P'_i)$ to $\mathcal{A}$.

Finally, $\mathcal{A}'$ parse $\sigma_* = (r_*, z_*, P_*)$ and returns P_*.

If there exists $i \in \{1, \cdots, n\}$ such that $\mathsf{pk}_i = g^{x'/x_i}$, then $\mathcal{B}$ returns $x = x'/x_i$, else it returns $\perp$.

Analyze: First note that the experiment n-unf is perfectly simulated for $\mathcal{A}$, then it returns (L_*, σ_*, m_*) such that for $\sigma_* = (r_*, z_*, P_*)$ and $h_* = H(m_*||r_*)$, we have $\mathsf{Pr}[\mathsf{LEverif}_{|L_*|}(\{(h_*, z_*, g_*, \mathsf{pk}_l)\}_{\mathsf{pk}_l \in L_*}, P_*) = 1] \geq \lambda(k)$ and $L_* \subset \{\mathsf{pk}_i\}_{1 \leq i \leq n}$. We deduce that $\mathcal{A}'$ returns a proof P_* such that $\mathsf{Pr}[\mathsf{LEverif}_{|L_*|}(\{(h_*, z_*, g_*, \mathsf{pk}_l)\}_{\mathsf{pk}_l \in L_*}, P_*) = 1] = \lambda(k)$, then $\mathcal{E}^{\mathcal{A}'(\{\mathsf{pk}_i\}_{1 \leq i \leq n})}(k)$ returns the discrete logarithm x' of one of the public keys in $\{\mathsf{pk}_i\}_{1 \leq i \leq n}$ with probability at lest $\lambda(k)$. Suppose that $\mathcal{A}'$ returns a valid proof, Since for all i, $\mathsf{pk}_i = y^{x_i}$, and since there exists j such that $\mathsf{pk}_j = g^{x'}$, then the discrete logarithm of y is x'/x_j. We deduce that $\mathcal{B}$ returns the discrete logarithm of y with probability at lest $\lambda(k)$. □

Lemma 2. EVeR *is n-ano secure for any polynomially bounded n under the DDH assumption in the random oracle model.*

Proof. Let the n-ano_ψ be the same experiment as n-ano except that the oracle $\mathsf{LRSO_b}$ can be called at most ψ times. We prove the two following claims:

Claim 1 If $\exists \mathcal{A} \in \text{POLY}(k)$ such that $\lambda_1(k) = \mathsf{Adv}^{n\text{-}\mathsf{ano}_1}_{\text{EVeR},\mathcal{A}}(k)$ is non-negligible, then $\exists \mathcal{B} \in \text{POLY}(k)$ that breaks the DDH assumption with non-negligible probability.

Claim 2 Let $\psi \geq 1$ be, suppose that $\epsilon(k) = \mathsf{Adv}^{n\text{-}\mathsf{ano}_\psi}_{\text{EVeR},\mathcal{A}}(k)$ is negligible. Then, if $\exists \mathcal{A} \in \text{POLY}(k)$ such that $\lambda_{\psi+1}(k) = \mathsf{Adv}^{n\text{-}\mathsf{ano}_{\psi+1}}_{\text{EVeR},\mathcal{A}}(k)$ is non-negligible, then $\exists \mathcal{B} \in \text{POLY}(k)$ that breaks the DDH assumption with non-negligible probability.

This two claims imply that $\mathsf{Adv}^{n\text{-}\mathsf{ano}_\psi}_{\text{EVeR},\mathcal{A}}(k)$ is negligible for any n and any ψ that are polynomially bounded.

Proof of Claim 1: We show how to build the algorithm $\mathcal{B}$. It receives a DDH instance $((G, p, g), X, Y, Z)$ as input. It picks $d \xleftarrow{\$} \{1, \cdots, n\}$. For all $i \in \{1, \cdots, n\}$:

- if $i = d$ then it sets $\mathsf{pk}_i = X$
- else, it runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{V.Gen}(\mathsf{init})$ where $\mathsf{init} = (G, p, g, H)$.

$\mathcal{B}$ runs $(d_0, d_1) \leftarrow \mathcal{A}_1(\{\mathsf{pk}_i\}_{1 \leq i \leq n})$. During the experiment, $\mathcal{B}$ simulates the oracle for $\mathcal{A}$ as follows:

Random oracle $H(.)$**:** On the i^{th} input M_i, if $\exists j < i$ such that $M_j = M_i$ then it sets $u_j = u_i$. Else it picks $u_i \xleftarrow{\$} \mathbb{Z}_p^*$. Finally, it returns g^{u_i}.

Oracle $\mathsf{V.Sig}(\cdot, \cdot, \cdot)$**:** On the i^{th} input (L_i, l_i, m_i), it picks $r_i \xleftarrow{\$} \mathbb{Z}_p^*$. It computes $h_i = H(m_i||r_i)$ using the oracle $H(.)$, then there exists j such that $m'_i||r'_i = M_j$.
- If $l_i = d$ then it computes $z_i = X^{u_j}$ and it runs $P_i \leftarrow \mathsf{Sim}(\{(h_i, z_i, g, \mathsf{pk}_l)\}_{\mathsf{pk}_l \in L_i})$. It returns $\sigma_i = (r_i, z_i, P_i)$ to $\mathcal{A}$.
- Else it runs and returns $\sigma_i \leftarrow \mathsf{V.Sig}(L_i, \mathsf{sk}_{l_i}, m_i)$

Oracle $\mathsf{V.Proof}(\cdot, \cdot, \cdot, \cdot, \cdot)$**:** On the i^{th} input $(L'_i, m'_i, \sigma'_i, l'_i)$, it parses $\sigma'_i = (r'_i, z'_i, P'_i)$. It computes $h'_i = H(m'_i||r'_i)$ using the oracle $H(.)$, then there exists j such that $m'_i||r'_i = M_j$ It computes $\bar{z}_i = \mathsf{pk}_{l'_i}^{u_j}$ and it runs $P'_i \leftarrow \mathsf{Sim}((h'_i, \bar{z}_i, g, \mathsf{pk}_{l'_i})$. It returns $(\bar{z}_i, P'_i)$ to $\mathcal{A}$.

$\mathcal{B}$ runs $b_* \leftarrow \mathcal{A}_2(\{\mathsf{pk}_i\}_{1 \leq i \leq n})$. During the experiment, $\mathcal{B}$ simulates the oracle $\mathsf{V.Sig}(\cdot, \cdot, \cdot)$ as in the first phase. It simlulates the three other oracles as follows:

Oracle $\mathsf{LRSO_b}(d_0, d_1, \cdot, \cdot)$**:** On input (m'', L''), it picks $r'' \xleftarrow{\$} \mathbb{Z}_p^*$. If $\exists i$ such that $r_i = r''$ then $\mathcal{B}$ aborts the experiment and returns $b'_* \xleftarrow{\$} \{0,1\}$, else it runs $P'' \leftarrow \mathsf{Sim}(\{(Y, Z, g, \mathsf{pk}_l)\}_{\mathsf{pk}_l \in L''})$ and returns (r'', Z, P'') to $\mathcal{A}$.

Oracle V.Proof$(\cdot, \cdot, \cdot, \cdot, \cdot)$**:** On the i^{th} input $(L'_i, m'_i, \sigma'_i, l'_i)$, if $\mathsf{LRSO_b}$ have been already called and $\sigma'_i = \sigma''$ and ($l'_i = d_0$ or $l'_i = d_1$) then it returns $\perp$ to $\mathcal{A}$. Else, it process as in first phase.

Random oracle $H(.)$**:** On the i^{th} input M_i, if $\mathsf{LRSO_b}$ have been already called and $M_i = (r''||m'')$ then it returns Y to $\mathcal{A}$. Else, it process as in first phase.

Let b' be the bit that verifies $d_{b'} = d$. If $b' = b_*$ then $\mathcal{B}$ returns $b'_* = 1$, else $b'_* = 0$.

Analyze: Let q be the number of queries asked to $\mathsf{V.Sig}(\cdot, \cdot, \cdot)$ and let E be the event "$\mathcal{B}$ does not aborts the experiment of $\mathcal{A}$. We have:

$$\begin{aligned}
\mathsf{Pr}[\neg E] &= \mathsf{Pr}[(\exists i, r_i = r'') \vee (d_0 \neq d \wedge d_1 \neq d)] \\
&\leq \mathsf{Pr}[(\exists i, r_i = r'')] + \mathsf{Pr}[(d_0 \neq d \wedge d_1 \neq d)] \\
&\leq \sum_{i=1}^{q} \mathsf{Pr}[r_i = r''] + \mathsf{Pr}[(d_0 \neq d \wedge d_1 \neq d)] \\
&\leq \frac{q}{|G|} + \frac{1}{n}
\end{aligned}$$

We deduce that:

$$\mathsf{Pr}[E] \geq 1 - \left(\frac{q}{|G|} + \frac{1}{n}\right) \geq \frac{n-1}{n} - \frac{q}{|G|} \geq \frac{1}{n} - \frac{q}{|G|}$$

Let α, β be two elements of G such that $X = g^\alpha$ and $Y = g^\beta$. Let b be the solution to the DDH instance, *i.e.* $b = 1$ iff $Z = g^{\alpha \cdot \beta}$. We compute the probability that $\mathcal{B}$ wins its DDH experiment:

$$\begin{aligned}
\mathsf{Pr}[b'_* = b] &= \mathsf{Pr}[E] \cdot \mathsf{Pr}[b'_* = b|E] + (1 - \mathsf{Pr}[E]) \cdot \mathsf{Pr}[b'_* = b|\neg E] \\
&= \mathsf{Pr}[E] \cdot (\mathsf{Pr}[b'_* = b|E] - \mathsf{Pr}[b'_* = b|\neg E]) + \mathsf{Pr}[b'_* = b|\neg E] \\
&= \mathsf{Pr}[E] \cdot (\mathsf{Pr}[b'_* = b|E] - \frac{1}{2}) + \frac{1}{2} \\
&= \mathsf{Pr}[E] \cdot (\mathsf{Pr}[Z = g^{\alpha\cdot\beta}] \cdot \mathsf{Pr}[b'_* = b|E \wedge (Z = g^{\alpha\cdot\beta})] \\
&\quad + \mathsf{Pr}[Z \neq g^{\alpha\cdot\beta}] \cdot \mathsf{Pr}[b'_* = b|E \wedge (Z \neq g^{\alpha\cdot\beta})] - \frac{1}{2}) + \frac{1}{2} \\
&= \mathsf{Pr}[E] \cdot (\frac{1}{2} \cdot (\frac{1}{2} \pm \lambda_1(k)) + \frac{1}{2} \cdot \frac{1}{2} - \frac{1}{2}) + \frac{1}{2} \\
&= \pm\lambda_1(k) \cdot \frac{\mathsf{Pr}[E]}{2} + \frac{1}{2}
\end{aligned}$$

Finally, we deduce the advantage of $\mathcal{B}$ against the DDH problem:

$$\left|\mathsf{Pr}[b'_* = b] - \frac{1}{2}\right| = \lambda_1(k) \cdot \frac{\mathsf{Pr}[E]}{2} = \lambda_1(k) \cdot \left(\frac{1}{2 \cdot n} - \frac{q}{2 \cdot |G|}\right)$$

This advantage is non-negligible, which conclude the proof of Claim 1.

Proof of Claim 2: We show how to build the algorithm $\mathcal{B}$. It runs the same reduction as in claim 1, except that the algorithm $\mathcal{B}$ simulates the oracles $\mathsf{LRSO_b}(d_0, d_1, \cdot, \cdot)$ and $\mathsf{V.Proof}(\cdot, \cdot, \cdot, \cdot, \cdot)$ as follows during the second phase of the $\mathcal{A}$ experiment:

Oracle $\mathsf{LRSO_b}(d_0, d_1, \cdot, \cdot)$**:** On the i^{th} input (m''_i, L''_i), if $i = 0$ then this oracle is defined as in the reduction of the Claim 1. Else it runs the oracle $\mathsf{V.Sig}(\cdot, \cdot, \cdot)$ on the input (m''_i, d, L''_i) and returns the resulted signature σ''_i to $\mathcal{A}$.

Oracle V.Proof$(\cdot, \cdot, \cdot, \cdot, \cdot)$**:** On the i^{th} input $(L'_i, m'_i, \sigma'_i, l'_i)$, if $\mathsf{LRSO_b}$ have been already called and $\exists j$ such that $\sigma'_i = \sigma''_j$ and ($l'_i = d_0$ or $l'_i = d_1$) then it returns $\perp$ to $\mathcal{A}$. Else, it process as in the reduction of Claim 1.

Analyze: Let q be the number of queries asked to $\mathsf{V.Sig}(\cdot, \cdot, \cdot)$ and let E be the event "$\mathcal{B}$ does not aborts the experiment of $\mathcal{A}$. As in Claim 1, we have:

$$\mathsf{Pr}[E] \geq \frac{1}{n} - \frac{q}{|G|}$$

Let α, β be two elements of G such that $X = g^\alpha$ and $Y = g^\beta$. Let b be the solution to the DDH instance, *i.e.* $b = 1$ iff $Z = g^{\alpha \cdot \beta}$. We compute the probability that $\mathcal{B}$ wins its DDH experiment:

$$\begin{aligned}
\mathsf{Pr}[b'_* = b] &= \mathsf{Pr}[E] \cdot \mathsf{Pr}[b'_* = b|E] + (1 - \mathsf{Pr}[E]) \cdot \mathsf{Pr}[b'_* = b|\neg E] \\
&= \mathsf{Pr}[E] \cdot (\mathsf{Pr}[Z = g^{\alpha \cdot \beta}] \cdot \mathsf{Pr}[b'_* = b|E \wedge (Z = g^{\alpha \cdot \beta})] \\
&\quad + \mathsf{Pr}[Z \neq g^{\alpha \cdot \beta}] \cdot \mathsf{Pr}[b'_* = b|E \wedge (Z \neq g^{\alpha \cdot \beta})] - \frac{1}{2}) + \frac{1}{2} \\
&= \mathsf{Pr}[E] \cdot (\frac{1}{2} \cdot (\frac{1}{2} \pm \lambda(k)) + (\frac{1}{2} \pm \epsilon_1(k)) - \frac{1}{2}) + \frac{1}{2} \\
&= (\pm\lambda(k) \pm \epsilon(k)) \cdot \frac{\mathsf{Pr}[E]}{2} + \frac{1}{2}
\end{aligned}$$

Finally, we deduce the advantage of $\mathcal{B}$ against the DDH problem:

$$\begin{aligned}\left|\mathsf{Pr}[b'_* = b] - \frac{1}{2}\right| &= |\pm\lambda(k) \pm \epsilon(k)| \cdot \frac{\mathsf{Pr}[E]}{2} \\ &\geq (\lambda(k) - \epsilon(k)) \cdot \left(\frac{1}{2\cdot n} - \frac{q}{2\cdot|G|}\right) \\ &\geq \lambda(k) \cdot \frac{1}{2\cdot n} - \left(\frac{q\cdot\lambda(k)}{2\cdot|G|}\right) - \epsilon(k) \cdot \left(\frac{1}{2\cdot n} - \frac{q}{2\cdot|G|}\right) \\ &\geq \lambda(k) \cdot \frac{1}{2\cdot n} - \left(\frac{q\cdot\lambda(k)}{2\cdot|G|}\right) - \frac{\epsilon(k)}{2\cdot n}\end{aligned}$$

This advantage is non-negligible, which conclude the proof of Claim 2 and the proof of the lemma. □

Lemma 3. EVeR *is n-acc secure for any polynomially bounded n under the DL assumption in the random oracle model.*

Proof. We first recall that since LogEq_n is valid, then there exists a polynomial time extractor $\mathcal{E}$ and a polynomial time simulator Sim for LogEq_n. Suppose that there exists an adversary $\mathcal{A} \in \text{POLY}(k)$ such the advantage $\lambda(k) = \mathsf{Pr}[\mathsf{Exp}^{n\text{-acc}}_{\text{EVeR},\mathcal{A}}(k) = 1]$ is non-negligible. We show how to build an algorithm $\mathcal{B} \in \text{POLY}(k)$ that solve the DL problem with non-negligible probability.

$\mathcal{B}$ description: For all $i \in \{1, \cdots, n\}$, it picks $x_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\mathsf{pk}_i = Y^{x_i}$. $\mathcal{B}$ runs $x' \leftarrow \mathcal{E}^{\mathcal{A}'(\{\mathsf{pk}_i\}_{1\leq i\leq n})}(k)$ where $\mathcal{A}'$ is the following algorithm:

Algorithm $\mathcal{A}'(\{\mathbf{pk}_i\}_{1\leq i\leq n})$**:** It runs $(L_*, m_*, \sigma_*, \mathsf{pk}_*, \pi_*) \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{1\leq i\leq n})$. It simulates the oracles to $\mathcal{A}$ as follows:

- **Random oracle** $H(.)$**:** On the i^{th} input M_i, if $\exists j < i$ such that $M_j = M_i$ then it sets $u_j = u_i$. Else it picks $u_i \xleftarrow{\$} \mathbb{Z}_p^*$. Finally, it returns g^{u_i}.
- **Oracle** $\mathbf{V.Sig}(\cdot,\cdot,\cdot)$**:** On the i^{th} input (L_i, l_i, m_i), it picks $r_i \xleftarrow{\$} \mathbb{Z}_p^*$. It computes $h_i = H(m_i||r_i)$ using the oracle $H(.)$, then there exists j such that $m'_i||r'_i = M_j$. It computes $z_i = X^{u_j}$ and it runs $P_i \leftarrow \mathsf{Sim}(\{(h_i, z_i, g, \mathsf{pk}_l)\}_{\mathsf{pk}_l\in L_i})$. It returns $\sigma_i = (r_i, z_i, P_i)$ to $\mathcal{A}$.
- **Oracle** $\mathbf{V.Proof}(\cdot,\cdot,\cdot,\cdot,\cdot)$**:** On the i^{th} input $(L'_i, m'_i, \sigma'_i, l'_i)$, it parses $\sigma'_i = (r'_i, z'_i, P'_i)$. It computes $h'_i = H(m'_i||r'_i)$ using the oracle $H(.)$, then there exists j such that $m'_i||r'_i = M_j$. It computes $\bar{z}_i = \mathsf{pk}_{l'_i}^{u_j}$ and it runs $P'_i \leftarrow \mathsf{Sim}((h'_i, \bar{z}_i, g, \mathsf{pk}_{l'_i})$. It returns $(\bar{z}_i, \pi'_i)$ to $\mathcal{A}$.

Finally, $\mathcal{A}'$ compute $h_*(r_*||m_*)$ using the random oracle $H(\cdot)$, parses $\sigma_* = (r_*, z_*, P_*)$ and returns P_*.

If there exists $i \in \{1, \cdots, n\}$ such that $\mathsf{pk}_i = g^{x'/x_i}$, then $\mathcal{B}$ returns $x = x'/x_i$, else it returns $\perp$.

Analyze: We parse $\sigma_* = (r_*, z_*, P_*)$ and $\pi_* = (\bar{z}_*, P'_*)$. Suppose that $\mathcal{A}$ wins the experiment, then we have:

$$L_* \subseteq \{\mathsf{pk}_i\}_{1 \leq i \leq n} \cup \{\mathsf{pk}_*\} \tag{1}$$

$$\mathsf{V.Ver}(L_*, \sigma_*, m_*) = 1 \tag{2}$$

$$\mathsf{V.Judge}(L_*, m_*, \sigma_*, \mathsf{pk}_*, \pi_*) = 0 \tag{3}$$

$$\forall\, i \in \{1, \ldots, q_S\}, \sigma_i \neq \sigma_* \tag{4}$$

Moreover, equation (4) implies that $\forall\ i \in \{1, \ldots, q_S\}, P_i \neq P_*$, then P_i was not generated by the simulator Sim. We deduce the following equation from (2):

$$\mathsf{LEverif}_{|L_*|}(\{(h_*, z_*, g, \mathsf{pk}_l)\}_{\mathsf{pk}_l \in L_*}, P_*) = 1 \tag{5}$$

Thus $\mathcal{A}$ returns a valid proof with non negligible probability $\lambda(k)$. Since $\mathcal{E}$ is an extractor for LogEq_n, it implies that:

$$\mathsf{Pr}[\exists\ \mathsf{pk} \in L_*, x' = \log_g(\mathsf{pk}) = \log_{h_*}(z_*)] \geq \lambda(k) \tag{6}$$

We deduce the following equation from (3):

$$\bar{z}_* \neq z_* \tag{7}$$

$$\mathsf{LEverif}_1(\{(h_*, \bar{z}_*, g, \mathsf{pk}_*)\}, P'_*) = 1 \tag{8}$$

Since LogEq_n is sound, we deduce there exists a negligible function ϵ such that:

$$\mathsf{Pr}[\log_g(\mathsf{pk}_*) = \log_{h_*}(\bar{z}_*)] \geq 1 - \epsilon(k) \tag{9}$$

$$\Rightarrow \mathsf{Pr}[\log_g(\mathsf{pk}_*) \neq \log_{h_*}(z_*)] \geq 1 - \epsilon(k) \tag{10}$$

$$\Rightarrow \mathsf{Pr}[\log_g(\mathsf{pk}_*) = \log_{h_*}(z_*)] \leq \epsilon(k) \tag{11}$$

Finally, from (1), (6) and (11) we deduce the probability that $\mathcal{B}$ wins the experiment:

$$\begin{aligned}
&\mathsf{Pr}[\exists\ \mathsf{pk} \in L_*, x' = \log_g(\mathsf{pk}) = \log_{h_*}(z_*)] \geq \lambda(k) \\
&\Leftrightarrow \mathsf{Pr}[\exists\ \mathsf{pk} \in L_* \backslash \{\mathsf{pk}_*\}, x' = \log_g(\mathsf{pk}) = \log_{h_*}(z_*)] + \mathsf{Pr}[x' = \log_g(\mathsf{pk}_*) = \log_{h_*}(z_*)] \geq \lambda(k) \\
&\Leftrightarrow \mathsf{Pr}[Y = g^x] + \mathsf{Pr}[x' = \log_g(\mathsf{pk}_*) = \log_{h_*}(z_*)] \geq \lambda(k) \\
&\Leftrightarrow \mathsf{Pr}[Y = g^x] \geq \lambda(k) - \mathsf{Pr}[\log_g(\mathsf{pk}_*) = \log_{h_*}(z_*)] \\
&\Leftrightarrow \mathsf{Pr}[Y = g^x] \geq \lambda(k) - \epsilon(k)
\end{aligned}$$

Since $\mathsf{Pr}[Y = g^x]$ is non negligible then $\mathcal{B}$ solve the DL problem with non-negligible probability. □

Lemma 4. EVeR *is n-non-usu-2 secure for any polynomially bounded n under the DL assumption in the random oracle model.*

Proof. We first recall that since LogEq_n is valid, then there exists a polynomial time extractor $\mathcal{E}$ and a polynomial time simulator Sim for LogEq_n. Suppose that there exists an adversary $\mathcal{A} \in \text{POLY}(k)$ such the advantage $\lambda(k) = \mathsf{Pr}[\mathsf{Exp}^{n\text{-non-usu-2}}_{\text{EVeR},\mathcal{A}}(k) = 1]$ is non-negligible. We show how to build an algorithm $\mathcal{B} \in \text{POLY}(k)$ that solve the DL problem with non-negligible probability.

$\mathcal{B}$ description: $\mathcal{B}$ sets $\mathsf{pk} = Y$. $\mathcal{B}$ runs $x \leftarrow \mathcal{E}^{\mathcal{A}'(\mathsf{pk})}(k)$ where $\mathcal{A}'$ is the following algorithm:

Algorithm $\mathcal{A}'(\mathbf{pk})$: It runs $(L_*, m_*, \sigma_*) \leftarrow \mathcal{A}(\mathsf{pk})$. It simulates the oracles to $\mathcal{A}$ as in the reduction of the previous proof. Finally, $\mathcal{A}'$ compute $h_*(r_*||m_*)$ using the random oracle $H(\cdot)$, parses $\sigma_* = (r_*, z_*, P_*)$ and returns P_*.

Finally, $\mathcal{B}$ returns x.

Analyze: We parse $\sigma_* = (r_*, z_*, P_*)$. Suppose that $\mathcal{A}$ wins the experiment, then we have, for any $\pi_* \leftarrow \mathsf{V.Proof}(L_*, m_*, \sigma_*, \mathsf{pk}, \mathsf{sk})$ where $\pi_* = (\bar{z}_*, P'_*)$:

$$\mathsf{V.Ver}(L_*, \sigma_*, m_*) = 1 \tag{12}$$

$$\mathsf{V.Judge}(L_*, m_*, \sigma_*, \mathsf{pk}, \pi_*) = 1 \tag{13}$$

$$\forall\, i \in \{1, \ldots, q_S\}, \sigma_i \neq \sigma_* \tag{14}$$

Moreover, equation (4) implies that $\forall\ i \in \{1, \ldots, q_S\}, P_i \neq P_*$, then P_i was not generated by the simulator Sim. We deduce the following equation from (12):

$$\mathsf{LEverif}_{|L_*|}(\{(h_*, z_*, g, \mathsf{pk}_l)\}_{\mathsf{pk}_l \in L_*}, P_*) = 1 \tag{15}$$

Thus $\mathcal{A}$ returns a valid proof with non negligible probability $\lambda(k)$. Since $\mathcal{E}$ is an extractor for LogEq_n, it implies that:

$$\mathsf{Pr}[\exists\, \mathsf{pk}_l \in L_*, x = \log_g(\mathsf{pk}_l) = \log_{h_*}(z_*)] \geq \lambda(k) \tag{16}$$

We deduce the following equation from (13):

$$\bar{z}_* = z_* \tag{17}$$

$$\mathsf{LEverif}_1(\{(h_*, z_*, g, \mathsf{pk})\}, P'_*) = 1 \tag{18}$$

Since LogEq_n is sound, we deduce there exists a negligible function ϵ such that:

$$\mathsf{Pr}[\log_g(\mathsf{pk}) = \log_{h_*}(z_*)] \geq 1 - \epsilon(k) \tag{19}$$

$$\Leftrightarrow \mathsf{Pr}[\log_g(\mathsf{pk}) \neq \log_{h_*}(z_*)] \leq \epsilon(k) \tag{20}$$

Finally, from (16) and (20) we deduce the probability that $\mathcal{B}$ wins the experiment:

$$\begin{aligned}
&\mathsf{Pr}[\exists\, \mathsf{pk}_l \in L_*, x = \log_g(\mathsf{pk}_l) = \log_{h_*}(z_*)] \geq \lambda(k) \\
&\Leftrightarrow \mathsf{Pr}[x = \log_g(\mathsf{pk}) = \log_{h_*}(z_*)] + \mathsf{Pr}[\exists\, \mathsf{pk}_l \in L_* \backslash \{\mathsf{pk}\}, x = \log_g(\mathsf{pk}_l) = \log_{h_*}(z_*)] \geq \lambda(k) \\
&\Leftrightarrow \mathsf{Pr}[x = \log_g(\mathsf{pk}) = \log_{h_*}(z_*)] \geq \lambda(k) - \mathsf{Pr}[\exists\, \mathsf{pk}_l \in L_* \backslash \{\mathsf{pk}\}, x = \log_g(\mathsf{pk}_l) = \log_{h_*}(z_*)] \\
&\Leftrightarrow \mathsf{Pr}[Y = g^x] \geq \lambda(k) - \mathsf{Pr}[\log_g(\mathsf{pk}) \neq \log_{h_*}(z_*)] \\
&\Leftrightarrow \mathsf{Pr}[Y = g^x] \geq \lambda(k) - \epsilon(k)
\end{aligned}$$

Since $\mathsf{Pr}[Y = g^x]$ is non negligible then $\mathcal{B}$ solve the DL problem with non-negligible probability. □

D Security proofs of GUSS

Lemma 5. *If D is unf secure then* GUSS *is immut secure.*

Proof. Suppose that there exists an adversary $\mathcal{A} \in \text{POLY}(k)$ such the advantage $\lambda(k) = \mathsf{Pr}[\mathsf{Exp}^{\mathsf{immut}}_{\text{GUSS},\mathcal{A}}(k) = 1]$ is non-negligible. We show how to build an algorithm $\mathcal{B} \in \text{POLY}(k)$ such that $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unf}}_{D,\mathcal{B}}(k) = 1]$ is non-negligible.

$\mathcal{B}$ construction: $\mathcal{B}$ receives the public key pk_d as input. It runs $\mathsf{init}_v \leftarrow \mathsf{V.Init}(1^k)$ and $(\mathsf{pk}_v, \mathsf{sk}_v) \leftarrow \mathsf{V.Gen}(\mathsf{init}_v)$. It sets $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$ and runs $(\mathsf{spk}_*, m_*, \sigma_*) \leftarrow \mathcal{A}(\mathsf{pk})$. During the experiment, $\mathcal{B}$ simulates the two oracles $\mathsf{Sig}(., \mathsf{sk}, ., .)$ and $\mathsf{SiProof}(\mathsf{sk}, ., ., .)$ to $\mathcal{A}$ as follows:

$\mathsf{Sig}(., \mathsf{sk}, ., .)$**:** On the i^{th} input $(m_i, \text{ADM}_i, \mathsf{spk}_i)$, $\mathcal{B}$ first computes the fixed message part $M \leftarrow \text{FIX}_{\text{ADM}_i}(m_i)$ and sends $(M_i||\text{ADM}_i||\mathsf{pk}||\mathsf{spk}_i)$ to the oracle $\mathsf{D.Sig}(\mathsf{sk}_d, \cdot)$ and receives the signature $\sigma_{i,1}$. It runs $\sigma_2 \leftarrow \mathsf{V.Sig}(\{\mathsf{pk}_v, \mathsf{spk}_i\}, \mathsf{sk}_v, (\sigma_{i,1}||m_i)))$. It returns $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \text{ADM}_i)$.

$\mathsf{SiProof}(\mathsf{sk}, ., ., .)$**:** On the i^{th} input $(m'_i, \sigma'_i, \mathsf{spk}'_i)$, It parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \text{ADM}'_i)$. It runs $\pi'_{\mathsf{si},i} \leftarrow \mathsf{V.Proof}(\{\mathsf{pk}_v, \mathsf{spk}'_i\}, (m'_i||\sigma'_{i,1}), \sigma'_{i,2}, \mathsf{pk}_v, \mathsf{sk}_v)$ and returns it.

Finally, $\mathcal{B}$ parses $\sigma_* = (\sigma_{1,*}, \sigma_{2,*}, \text{ADM}_*)$, $M_* \leftarrow \text{FIX}_{\text{ADM}_*}(m_*)$ and returns the couple $((M_*||\text{ADM}_*||\mathsf{pk}||\mathsf{spk}_*), \sigma_{1,*})$.

analyze We show the if $\mathcal{A}$ wins its experiment, then $\mathcal{B}$ also wins its experiment. Suppose that $\mathcal{A}$ wins its experiment, then the following equations holds:

$$\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*) = 1 \tag{21}$$

$$\forall\, i \in \{1, \ldots, q_{\mathsf{Sig}}\}, (\mathsf{spk}_* \neq \mathsf{spk}_i) \text{ or } (\text{FIX}_{\text{ADM}_*}(m_*) \neq \text{FIX}_{\text{ADM}_i}(m_i)) \tag{22}$$

(21) implies the following equation:

$$\mathsf{D.Ver}(\mathsf{pk}_d, (M_*||\text{ADM}_*||\mathsf{pk}||\mathsf{spk}_*), \sigma_{1,*}) = 1$$

Moreover, (22) implies that:

$$\forall\, i \in \{1, \ldots, q_{\mathsf{Sig}}\}, (M_*||\text{ADM}_*||\mathsf{pk}||\mathsf{spk}_*) \neq (M_i||\text{ADM}_i||\mathsf{pk}||\mathsf{spk}_i)$$

We deduce that $\mathcal{B}$ never sends the message $(M_*||\text{ADM}_*||\mathsf{pk}||\mathsf{spk}_*)$ to the oracle $\mathsf{Sig}(., \mathsf{sk}, ., .)$. Moreover, we deduce that if $\mathcal{A}$ wins its experiment, then $\mathcal{B}$ wins its experiment, thus $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unf}}_{D,\mathcal{B}}(k) = 1] \geq \lambda(k)$ □

Lemma 6. *If V is 2-ano secure then* GUSS *is trans secure.*

Proof. Suppose that there exists an adversary $\mathcal{A} \in \text{POLY}(k)$ such the advantage $\lambda(k) = \mathsf{Pr}[\mathsf{Exp}^{\mathsf{trans}}_{\text{GUSS},\mathcal{A}}(k) = 1]$ is non-negligible. We show how to build an algorithm $\mathcal{B} \in \text{POLY}(k)$ such that $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{2\text{-}ano}}_{V,\mathcal{B}}(k) = 1]$ is non-negligible.

$\mathcal{B}$ construction: $\mathcal{B}_1$ receives $(\mathsf{spk}, \mathsf{pk}_v)$ as input, and returns $(1, 2)$. $\mathcal{B}_2$ runs $(\mathsf{pk}_d, \mathsf{sk}_d) \leftarrow \mathsf{SiGen}(\mathsf{D.Init}(1^k))$ and sets $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$. It runs $b' \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{spk})$ and returns b'. During the experiment, $\mathcal{B}_2$ simulates the oracles to $\mathcal{A}$ as follows:

$\mathsf{Sig}(., \mathsf{sk}, ., .)$**:** On the i^{th} input $(m_i, \text{ADM}_i, \mathsf{spk}_i)$, $\mathcal{B}_2$ first computes the fixed message part $M_i \leftarrow \text{FIX}_{\text{ADM}_i}(m_i)$ and runs $\sigma_{i,1} \leftarrow \mathsf{D.Sig}(\mathsf{sk}_d, (M_i||\text{ADM}_i||\mathsf{pk}||\mathsf{spk}_i))$ and it sends $(\{\mathsf{pk}_v, \mathsf{spk}_i\}, 1, (m_i||\sigma_{i,1}))$ to the oracle $\mathsf{V.Sig}(., ., .)$ that returns the signature $\sigma_{i,2}$. $\mathcal{B}_2$ returns $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \text{ADM}_i)$ to $\mathcal{A}$.

$\mathsf{San}(., ., ., ., \mathsf{ssk})$**:** On the i^{th} input $(m'_i, \text{MOD}'_i, \sigma'_i, \mathsf{pk}'_i)$, it parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \text{ADM}'_i)$ and $\mathsf{pk}'_i = (\mathsf{pk}'_{d,i}, \mathsf{pk}'_{v,i})$. This algorithm first computes the modified message $\bar{m}'_i \leftarrow \text{MOD}'_i(m'_i)$ and it sends $(\{\mathsf{pk}'_{v,i}, \mathsf{spk}\}, 2, (\bar{m}'_i||\sigma'_{i,1}))$ to the oracle $\mathsf{V.Sig}(., ., .)$ that returns the signature $\bar{\sigma}'_{i,2}$. $\mathcal{B}_2$ returns $\bar{\sigma}'_i = (\sigma'_{i,1}, \bar{\sigma}'_{i,2}, \text{ADM}'_i)$ to $\mathcal{A}$.

$\mathsf{SiProof}(\mathsf{sk}, ., ., .)$**:** On the i^{th} input $(m''_i, \sigma''_i, \mathsf{spk}''_i)$, $\mathcal{B}$ parses $\sigma''_i = (\sigma''_{i,1}, \sigma''_{i,2}, \text{ADM}''_i)$. It sends $(\{\mathsf{pk}_v, \mathsf{spk}''_i\}, (m''_i||\sigma''_{i,1}), \sigma''_{i,2}, \mathsf{pk}_v, 1)$ to the oracle $\mathsf{V.Proof}(., ., ., ., .)$ that returns the proof $\pi''_{\mathsf{si},i}$. Finally, $\mathcal{B}$ returns $\pi''_{\mathsf{si},i}$.

$\mathsf{SaProof}(\mathsf{ssk}, ., ., .)$**:** On the i^{th} input $(m'''_i, \sigma'''_i, \mathsf{pk}'''_i)$, $\mathcal{B}$ parses $\sigma'''_i = (\sigma'''_{i,1}, \sigma'''_{i,2}, \text{ADM}'''_i)$ and $\mathsf{pk}'''_i = (\mathsf{pk}'''_{d,i}, \mathsf{pk}'''_{v,i})$. It sends $(\{\mathsf{pk}'''_{v,i}, \mathsf{spk}\}, (m'''_i||\sigma'''_{i,1}), \sigma'''_{i,2}, \mathsf{spk}, 2)$ to the oracle $\mathsf{V.Proof}(., ., ., ., .)$ that returns the proof $\pi'''_{\mathsf{sa},i}$. Finally, $\mathcal{B}$ returns $\pi'''_{\mathsf{sa},i}$.

$\mathsf{Sa/Si}(b, \mathsf{pk}, \mathsf{spk}, \mathsf{sk}, \mathsf{ssk}, ., ., .)$**:** On the i^{th} input $(\tilde{m}_i, \tilde{\text{ADM}}_i, \tilde{\text{MOD}}_i)$, if $\tilde{\text{ADM}}_i(\tilde{\text{MOD}}_i) = 0$, $\mathcal{B}_2$ returns $\perp$. Else $\mathcal{B}_2$ computes the fixed message part $\tilde{M}_i \leftarrow \text{FIX}_{\tilde{\text{ADM}}_i}(\tilde{m}_i)$. It runs $\tilde{\sigma}_{i,1} \leftarrow \mathsf{D.Sig}(\mathsf{sk}_d, (\tilde{M}_i||\tilde{\text{ADM}}_i||\mathsf{pk}||\mathsf{spk}))$ and it sends $((\tilde{\text{MOD}}(\tilde{m}_i)||\tilde{\sigma}_{i,1}), \{\mathsf{pk}_v, \mathsf{skp}\})$ to the oracle $\mathsf{LRSO_b}(1, 2, \cdot, \cdot)$ that returns the signature $\tilde{\sigma}_{i,2}$. $\mathcal{B}_2$ returns $\tilde{\sigma}_i = (\tilde{\sigma}_{i,1}, \tilde{\sigma}_{i,2}, \tilde{\text{ADM}}_i)$ to $\mathcal{A}$.

analyze: Suppose that $\mathcal{A}$ wins its experiment, then $b = b'$ and:

$$S_{\mathsf{Sa/Si}} \cap (S_{\mathsf{SiProof}} \cup S_{\mathsf{SaProof}}) = \emptyset$$

where $S_{\mathsf{Sa/Si}}$ (resp. S_{SiProof} and S_{SaProof}) corresponds to the set of all signatures outputted by the oracle $\mathsf{Sa/Si}$ (resp. sending to the oracles $\mathsf{SiProof}$ and $\mathsf{SaProof}$). It implies that the messages sending to the oracle $\mathsf{V.Proof}(., ., ., ., .)$ was not already signed by $\mathsf{LRSO_b}(1, 2, \cdot, \cdot)$. More formally, we have:

$$\forall\, i, j \in \{1, \ldots, \max(q_S, q_P)\}, (\sigma_i \neq \sigma'_j)$$

where q_S (resp. q_P) is the number of calls to the oracle $\mathsf{V.Sig}(\cdot, \cdot, \cdot)$ (resp. $\mathsf{V.Proof}(\cdot, \cdot, \cdot, \cdot, \cdot)$). Finally, the probability that $\mathcal{B}$ wins its experiment is the same as the probability that $\mathcal{A}$ wins its experiments:

$$\mathsf{Pr}[\mathsf{Exp}^{\mathsf{2\text{-}ano}}_{V,\mathcal{B}}(k) = 1] \geq \lambda(k)$$

which conclude the proof. □

Lemma 7. *If D is unf secure then* GUSS *is unlink secure.*

Proof. Suppose that there exists an adversary $\mathcal{A} \in \text{POLY}(k)$ such the advantage $\lambda(k) = |\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unlink}}_{\text{GUSS},\mathcal{A}}(k) = 1] - 1/2|$ is non-negligible. We show how to build an algorithm $\mathcal{B} \in \text{POLY}(k)$ such that $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unf}}_{D,\mathcal{B}}(k) = 1]$ is non-negligible.

$\mathcal{B}$ construction: $\mathcal{B}_1$ receives (pk_d) as input, $\mathcal{B}$ runs $(\mathsf{pk}_v, \mathsf{sk}_v) \leftarrow \mathsf{SiGen}(\mathsf{V.Init}(1^k))$ and $(\mathsf{spk}, \mathsf{ssk}) \leftarrow \mathsf{SiGen}(\mathsf{V.Init}(1^k))$, and sets $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$. It chooses $b \xleftarrow{\$} \{0,1\}$ and runs $b' \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{spk})$. During the experiment, $\mathcal{B}_2$ simulates the oracles to $\mathcal{A}$ as follows:

$\mathsf{Sig}(.,\mathsf{sk},.,.)$**:** On the i^{th} input $(m_i, \textsc{Adm}_i, \mathsf{spk}_i)$, $\mathcal{B}$ first computes the fixed message part $M \leftarrow \textsc{Fix}_{\textsc{Adm}_i}(m_i)$ and sends $(M_i||\textsc{Adm}_i||\mathsf{pk}||\mathsf{spk}_i)$ to the oracle $\mathsf{D.Sig}(\mathsf{sk}_d, \cdot)$ and receives the signature $\sigma_{i,1}$. It runs $\sigma_2 \leftarrow \mathsf{V.Sig}(\{\mathsf{pk}_v, \mathsf{spk}_i\}, \mathsf{sk}_v, (\sigma_{i,1}||m_i)))$. It returns $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \textsc{Adm}_i)$.

$\mathsf{SiProof}(\mathsf{sk},.,.,.)$**:** On the i^{th} input $(m'_i, \sigma'_i, \mathsf{spk}'_i)$, It parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \textsc{Adm}'_i)$. It runs $\pi'_{\mathsf{si},i} \leftarrow \mathsf{V.Proof}(\{\mathsf{pk}_v, \mathsf{spk}'_i\}, (m'_i||\sigma'_{i,1}), \sigma'_{i,2}, \mathsf{pk}_v, \mathsf{sk}_v)$ and returns it.

$\mathsf{San}(.,.,.,.,\mathsf{ssk})$**:** On the i^{th} input $(m''_i, \textsc{Mod}''_i, \sigma''_i, \mathsf{pk}''_i)$, $\mathcal{B}$ runs $\bar{\sigma}''_i \leftarrow \mathsf{San}(m''_i, \textsc{Mod}''_i, \sigma''_i, \mathsf{pk}''_i, \mathsf{ssk})$ and returns $\bar{\sigma}''_i$ to $\mathcal{A}$.

$\mathsf{SaProof}(\mathsf{ssk},.,.,.)$**:** On the i^{th} input $(m'''_i, \sigma'''_i, \mathsf{pk}'''_i)$, $\mathcal{B}$ runs $\pi'''_{\mathsf{sa},i} \leftarrow \mathsf{SaProof}(\mathsf{ssk}, m'''_i, \sigma'''_i, \mathsf{pk}'''_i)$ and returns π_{sa} to $\mathcal{A}$.

$\mathsf{LRSan}(b,\mathsf{pk},\mathsf{ssk},.,.)$**:** On the i^{th} input $((\tilde{m}_{0,i}, \tilde{\textsc{Mod}}_{0,i}, \tilde{\sigma}_{0,i})(\tilde{m}_{1,i}, \tilde{\textsc{Mod}}_{1,i}, \tilde{\sigma}_{1,i}))$, if for $j \in \{0,1\}$, $\mathsf{Ver}(\tilde{m}_{j,i}, \tilde{\sigma}_{j,i}, \mathsf{pk}, \mathsf{spk}) = 1$ and $\tilde{\textsc{Adm}}_{0,i} = \tilde{\textsc{Adm}}_{1,i}$ and $\tilde{\textsc{Adm}}_{j,i}(\tilde{\textsc{Mod}}_{j,i}) = 1$ and $\tilde{\textsc{Mod}}_{0,i}(\tilde{m}_{0,i}) = \tilde{\textsc{Mod}}_{1,i}(\tilde{m}_{1,i})$, then this oracle returns $\tilde{\sigma}'_i = (\tilde{\sigma}'_{1,b,i}, \tilde{\sigma}'_{2,b,i}, \tilde{adm}'_b) \leftarrow \mathsf{San}(\tilde{m}_{b,i}, \tilde{\textsc{Mod}}_{b,i}, \tilde{\sigma}_{b,i}, \mathsf{pk}, \mathsf{ssk})$ to $\mathcal{A}$, else it returns $\perp$. Moreover, if for $j \in \{0,1\}$, $\mathsf{Ver}(\tilde{m}_{j,i}, \tilde{\sigma}_{j,i}, \mathsf{pk}, \mathsf{spk}) = 1$ and $\tilde{\textsc{Adm}}_{0,i} = \tilde{\textsc{Adm}}_{1,i}$ and $\tilde{\textsc{Adm}}_{j,i}(\tilde{\textsc{Mod}}_{j,i}) = 1$ and $\tilde{\textsc{Mod}}_{0,i}(\tilde{m}_{0,i}) = \tilde{\textsc{Mod}}_{1,i}(\tilde{m}_{1,i})$, and if $\exists x$ such that $\tilde{\sigma}_{x,i}$ was not already outputted by the oracle $\mathsf{D.Sig}(\mathsf{sk}_d, \cdot)$, then $\mathcal{B}$ returns $((\textsc{Fix}_{\bar{\textsc{Adm}}_{x,i}}(\tilde{m}_{x,i})||\bar{\textsc{Adm}}_{x,i}||\mathsf{pk}||\mathsf{spk}), \tilde{\sigma}_{x,i})$ to the challenger and aborts the experiment for $\mathcal{A}$.

If $\mathcal{B}$ has not already aborted the experiment, then it returns $\perp$.

analyze: First observe that, if for any $i \in \{1,\ldots,q\}$ where q is the number of queries to the oracle $\mathsf{LRSan}(b,\mathsf{pk},\mathsf{ssk},.,.)$, for $j \in \{0,1\}$, $\mathsf{Ver}(\tilde{m}_{j,i}, \tilde{\sigma}_{j,i}, \mathsf{pk}, \mathsf{spk}) = 1$ and $\tilde{\textsc{Adm}}_{0,i} = \tilde{\textsc{Adm}}_{1,i}$ and $\tilde{\textsc{Adm}}_{j,i}(\tilde{\textsc{Mod}}_{j,i}) = 1$ and $\tilde{\textsc{Mod}}_{0,i}(\tilde{m}_{0,i}) = \tilde{\textsc{Mod}}_{1,i}(\tilde{m}_{1,i})$, and $\tilde{\sigma}_{j,i}$ was already outputted by the oracle $\mathsf{D.Sig}(\mathsf{sk}_d, \cdot)$, then

$$\textsc{Fix}_{\bar{\textsc{Adm}}_{0,i}}(\tilde{m}_{0,i})||\bar{\textsc{Adm}}_{0,i}||\mathsf{pk}||\mathsf{spk} = \textsc{Fix}_{\bar{\textsc{Adm}}_{1,i}}(\tilde{m}_{1,i})||\bar{\textsc{Adm}}_{1,i}||\mathsf{pk}||\mathsf{spk}$$

Since D is deterministic, we deduce that $\tilde{\sigma}'_{1,b,i} = \tilde{\sigma}_{0,i} = \tilde{\sigma}_{1,i}$. On the other hand, the second parts of the outputted signature $\tilde{\sigma}'_{2,b,i}$ does not depend of b. Finally, $\tilde{\textsc{Adm}}'_{b,i} = \tilde{\textsc{Adm}}_{0,i} = \tilde{\textsc{Adm}}_{1,i}$, then $\tilde{\textsc{Adm}}'_{b,i}$ does not depend of b. We deduce that the outputted signature $\tilde{\sigma}'_{b,i}$ leaks no information about b. In this case, the best strategy of $\mathcal{A}$ to wins the experiment is to randomly guess the bit b'.

On the other hand, if there exists $i \in \{1,\ldots,q\}$ where q is the number of queries to the oracle $\mathsf{LRSan}(b,\mathsf{pk},\mathsf{ssk},.,.)$, for $j \in \{0,1\}$, $\mathsf{Ver}(\tilde{m}_{j,i}, \tilde{\sigma}_{j,i}, \mathsf{pk}, \mathsf{spk}) = 1$ and $\tilde{\textsc{Adm}}_{0,i} = \tilde{\textsc{Adm}}_{1,i}$ and $\tilde{\textsc{Adm}}_{j,i}(\tilde{\textsc{Mod}}_{j,i}) = 1$ and $\tilde{\textsc{Mod}}_{0,i}(\tilde{m}_{0,i}) = \tilde{\textsc{Mod}}_{1,i}(\tilde{m}_{1,i})$, and if $\exists x$ such that $\tilde{\sigma}_{x,i}$ was not already outputted by the oracle $\mathsf{D.Sig}(\mathsf{sk}_d, \cdot)$, then $\mathcal{B}$ returns $((\textsc{Fix}_{\bar{\textsc{Adm}}_{x,i}}(\tilde{m}_{x,i})||\bar{\textsc{Adm}}_{x,i}||\mathsf{pk}||\mathsf{spk}), \tilde{\sigma}_{x,i})$ to the challenger and wins its experiment. We denote this event by E. We have:

$$\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unf}}_{D,\mathcal{B}}(k) = 1] \geq \mathsf{Pr}[E]$$

On the other hand, we have:

$$\begin{aligned}\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unlink}}_{\mathrm{GUSS},\mathcal{A}}(k)=1] &= \mathsf{Pr}[E]\cdot\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unlink}}_{\mathrm{GUSS},\mathcal{A}}(k)=1|E] \\ &\quad + (1-\mathsf{Pr}[E])\cdot\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unlink}}_{\mathrm{GUSS},\mathcal{A}}(k)=1|\neg E] \\ &= \mathsf{Pr}[E]\cdot\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unlink}}_{\mathrm{GUSS},\mathcal{A}}(k)=1|E] + \frac{1}{2} - \frac{1}{2}\cdot\mathsf{Pr}[E]\end{aligned}$$

It implies that:

$$\mathsf{Pr}[E] = \frac{\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unlink}}_{\mathrm{GUSS},\mathcal{A}}(k)=1]-\frac{1}{2}}{\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unlink}}_{\mathrm{GUSS},\mathcal{A}}(k)=1|E]-\frac{1}{2}} = \frac{\pm\lambda(k)}{\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unlink}}_{\mathrm{GUSS},\mathcal{A}}(k)=1|E]-\frac{1}{2}} \geq \lambda(k)$$

Finally, we deduce that

$$\mathsf{Pr}[\mathsf{Exp}^{\mathsf{unf}}_{D,\mathcal{B}}(k)=1] \geq \lambda(k)$$

which conclude the proof □

Lemma 8. *If V is 1-acc secure then* GUSS *is SiAcc-1 secure.*

Proof. Suppose that there exists an adversary $\mathcal{A} \in \mathrm{POLY}(k)$ such the advantage $\lambda(k) = \mathsf{Pr}[\mathsf{Exp}^{\mathsf{SiAcc\text{-}1}}_{\mathrm{GUSS},\mathcal{A}}(k) = 1]$ is non-negligible. We show how to build an algorithm $\mathcal{B} \in \mathrm{POLY}(k)$ such that $\mathsf{Pr}[\mathsf{Exp}^{\text{1-acc}}_{V,\mathcal{B}}(k)=1]$ is non-negligible.

$\mathcal{B}$ construction: $\mathcal{B}$ receives (spk) as input, and runs runs $(\mathsf{pk}_*, m_*, \sigma_*, \pi_{si,*}) \leftarrow \mathcal{A}(\mathsf{spk})$. During the experiment, $\mathcal{B}$ simulates the oracles to $\mathcal{A}$ as follows:

$\mathsf{San}(.,.,.,.,\mathsf{ssk})$**:** On the i^{th} input $(m_i, \mathrm{MOD}_i, \sigma_i, \mathsf{pk}_i)$, it parses $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \mathrm{ADM}_i)$ and $\mathsf{pk}_i = (\mathsf{pk}_{d,i}, \mathsf{pk}_{v,i})$. This algorithm first computes the modified message $\bar{m}_i \leftarrow \mathrm{MOD}_i(m_i)$ and it sends $(\{\mathsf{pk}_{v,i}, \mathsf{spk}\}, 1, (\bar{m}_i || \sigma_{i,1}))$ to the oracle $\mathsf{V.Sig}(.,.,.)$ that returns the signature $\bar{\sigma}_{i,2}$. $\mathcal{B}_2$ returns $\bar{\sigma}_i = (\sigma_{i,1}, \bar{\sigma}_{i,2}, \mathrm{ADM}_i)$ to $\mathcal{A}$.

$\mathsf{SaProof}(\mathsf{ssk},.,.,.)$**:** On the i^{th} input $(m'_i, \sigma'_i, \mathsf{pk}'_i)$, $\mathcal{B}$ parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \mathrm{ADM}'_i)$ and $\mathsf{pk}'_i = (\mathsf{pk}'_{d,i}, \mathsf{pk}'_{v,i})$. It sends $(\{\mathsf{pk}'_{v,i}, \mathsf{spk}\}, (m'_i || \sigma'_{i,1}), \sigma'_{i,2}, \mathsf{spk}, 1)$ to the oracle $\mathsf{V.Proof}(.,.,.,.,.)$ that returns the proof $\pi'_{\mathsf{sa},i}$. Finally, $\mathcal{B}$ returns $\pi'_{\mathsf{sa},i}$ to $\mathcal{A}$.

Finally, $\mathcal{B}$ parses $\mathsf{pk}_* = (\mathsf{pk}_{d,*}, \mathsf{pk}_{v,*})$ and $\sigma_* = (\sigma_{1,*}, \sigma_{2,*}, \mathrm{ADM}_*)$ and returns $(\{\mathsf{spk}, \mathsf{pk}_{v,*}\}, m_* || \sigma_{1,*}, \sigma_{2,*}, \mathsf{pk}v, *, \pi_{si,*})$

analyze: Suppose that $\mathcal{A}$ wins its experiment, then:

$$\forall\, i \in \{1,\ldots,q_{\mathsf{San}}\}, (\sigma_* \neq \sigma'_i) \tag{23}$$

$$\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}) = 1 \tag{24}$$

$$\mathsf{SiJudge}(m_*, \sigma_*, \mathsf{pk}_*, \mathsf{spk}, \pi_{\mathsf{si},*}) = 0 \tag{25}$$

where q_{San} is the number of calls to the oracle $\mathsf{San}(.,.,.,.,\mathsf{ssk})$. First note that $\{\mathsf{spk}, \mathsf{pk}_{v,*}\} \subset \{\mathsf{spk}\} \cup \{\mathsf{pk}_{v,*}\}$. (23) implies that:

$$\forall\, i \in \{1,\ldots,q_S\}, \sigma_{*,2} \neq \bar{\sigma}_{i,2}$$

where q_S is the number of queries to $\mathsf{V.Sig}(.,.,.)$. Indeed, if $(\sigma_* \neq \sigma'_i)$ then $\sigma_{1,*} \neq \sigma_{1,i}$ or $\sigma_{2,*} \neq \sigma_{2,i}$ or $\text{ADM}_* \neq \text{ADM}_i$: if $\text{ADM}_* \neq \text{ADM}_i$ then $\sigma_{1,*} \neq \sigma_{1,i}$ because $\sigma_{1,*}$ (resp. $\sigma_{1,i}$) is a signature of ADM_* (resp. ADM_i). If $\sigma_{1,*} \neq \sigma_{1,i}$ then $\sigma_{2,*} \neq \sigma_{2,i}$ because $\sigma_{2,*}$ (resp. $\sigma_{2,i}$) is a signature of $\sigma_{1,*}$ (resp. $\sigma_{1,i}$). Finally, in all cases $\sigma_{*,2} \neq \bar{\sigma}_{i,2}$.

On the other hand, (24) implies that:

$$\mathsf{V.Ver}(\{\mathsf{spk}, \mathsf{pk}_{v,*}\}, \sigma_{2,*}, m_*||\sigma_{1,*}) = 1$$

Finally, (25) implies that:

$$\mathsf{V.Judge}(\{\mathsf{spk}, \mathsf{pk}_{v,*}\}, m_*||\sigma_{1,*}, \sigma_{2,*}, \mathsf{pk}_{v,*}, \pi_{si,*}) = 0$$

We deduce that the probability that $\mathcal{B}$ wins its experiment is the same as the probability that $\mathcal{A}$ wins its experiments:

$$\mathsf{Pr}[\mathsf{Exp}^{\text{1-acc}}_{V,\mathcal{B}}(k) = 1] \geq \lambda(k)$$

which conclude the proof. □

Lemma 9. *If V is 1-non-usu-2 secure then* GUSS *is SaAcc-1 secure.*

Proof. Suppose that there exists an adversary $\mathcal{A} \in \text{POLY}(k)$ such the advantage $\lambda(k) = \mathsf{Pr}[\mathsf{Exp}^{\text{SaAcc-1}}_{\text{GUSS},\mathcal{A}}(k) = 1]$ is non-negligible. We show how to build an algorithm $\mathcal{B} \in \text{POLY}(k)$ such that $\mathsf{Pr}[\mathsf{Exp}^{\text{1-non-usu-2}}_{V,\mathcal{B}}(k) = 1]$ is non-negligible.

$\mathcal{B}$ construction: $\mathcal{B}$ receives (pk_v) as input, it generates $(\mathsf{pk}_d, \mathsf{sk}_d) \leftarrow \mathsf{SiGen}(\mathsf{init}_d)$, sets $\mathsf{pk} = (\mathsf{pk}_d, \mathsf{pk}_v)$ and runs runs $(\mathsf{spk}_*, m_*, \sigma_*) \leftarrow \mathcal{A}(\mathsf{pk})$. During the experiment, $\mathcal{B}$ simulates the oracles to $\mathcal{A}$ as follows:

$\mathsf{Sig}(., \mathsf{sk}, ., .)$: On the i^{th} input $(m_i, \text{ADM}_i, \mathsf{spk}_i)$, $\mathcal{B}$ first computes the fixed message part $M_i \leftarrow \text{FIX}_{\text{ADM}_i}(m_i)$ and runs $\sigma_{i,1} \leftarrow \mathsf{D.Sig}(\mathsf{sk}_d, (M_i||\text{ADM}_i||\mathsf{pk}||\mathsf{spk}_i))$ and it sends $(\{\mathsf{pk}_v, \mathsf{spk}_i\}, 1, (m_i||\sigma_{i,1}))$ to the oracle $\mathsf{V.Sig}(.,.,.)$ that returns the signature $\sigma_{i,2}$. $\mathcal{B}$ returns $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \text{ADM}_i)$ to $\mathcal{A}$.

$\mathsf{SiProof}(\mathsf{sk}, ., ., .)$: On the i^{th} input $(m'_i, \sigma'_i, \mathsf{spk}'_i)$, $\mathcal{B}$ parses $\sigma'_i = (\sigma'_{i,1}, \sigma'_{i,2}, \text{ADM}'_i)$. It sends $(\{\mathsf{pk}_v, \mathsf{spk}'_i\}, (m'_i||\sigma'_{i,1}), \sigma'_{i,2}, \mathsf{pk}_v, 1)$ to the oracle $\mathsf{V.Proof}(.,.,.,.,.)$ that returns the proof $\pi'_{\mathsf{si},i}$. Finally, $\mathcal{B}$ returns $\pi'_{\mathsf{si},i}$.

Finally, $\mathcal{B}$ parses $\sigma_* = (\sigma_{1,*}, \sigma_{2,*}, \text{ADM}_*)$ and returns $(\{\mathsf{spk}_*, \mathsf{pk}_v\}, m_*||\sigma_{1,*}, \sigma_{2,*})$.

analyze: Suppose that $\mathcal{A}$ wins its experiment, then, for any $\pi_{\mathsf{si},*} \leftarrow \mathsf{SiProof}(\mathsf{sk}, m_*, \sigma_*, \mathsf{spk}_*)$:

$$\forall\, i \in \{1, \ldots, q_{\mathsf{Sig}}\}, (\sigma_* \neq \sigma'_i) \tag{26}$$

$$\mathsf{Ver}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*) = 1 \tag{27}$$

$$\mathsf{SaJudge}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*, \pi_{\mathsf{si},*}) \neq 1 \tag{28}$$

where q_{San} is the number of calls to the oracle $\mathsf{San}(.,.,.,.,\mathsf{ssk})$. First note that (26) implies that:

$$\forall\, i \in \{1, \ldots, q_S\}, \sigma_{*,2} \neq \bar{\sigma}_{i,2}$$

where q_S is the number of queries to $\mathsf{V.Sig}(.,.,.)$. Indeed, if $(\sigma_* \neq \sigma'_i)$ then $\sigma_{1,*} \neq \sigma_{1,i}$ or $\sigma_{2,*} \neq \sigma_{2,i}$ or $\text{ADM}_* \neq \text{ADM}_i$: if $\text{ADM}_* \neq \text{ADM}_i$ then $\sigma_{1,*} \neq \sigma_{1,i}$ because $\sigma_{1,*}$ (resp. $\sigma_{1,i}$) is a signature of ADM_* (resp. ADM_i). If $\sigma_{1,*} \neq \sigma_{1,i}$ then $\sigma_{2,*} \neq \sigma_{2,i}$ because $\sigma_{2,*}$ (resp. $\sigma_{2,i}$) is a signature of $\sigma_{1,*}$ (resp. $\sigma_{1,i}$). Finally, in all cases $\sigma_{*,2} \neq \bar{\sigma}_{i,2}$.

On the other hand, (27) implies that:

$$\mathsf{V.Ver}(\{\mathsf{spk}_*, \mathsf{pk}_v\}, \sigma_{2,*}, m_* || \sigma_{1,*}) = 1$$

Moreover, (28) implies that:

$$\mathsf{SaJudge}(m_*, \sigma_*, \mathsf{pk}, \mathsf{spk}_*, \pi_{\mathsf{si},*}) = 1$$

Indeed, $\pi_{\mathsf{si},*}$ cannot be equal to $\perp$ since it is computed by the proof algorithm from a valid signature. It implies that:

$$\mathsf{V.Judge}(\{\mathsf{spk}_*, \mathsf{pk}_v\}, m_* || \sigma_{1,*}, \sigma_{2,*}, \mathsf{pk}_v, \pi_{si,*}) = 0$$

Finally, note that since $\pi_{\mathsf{si},*} \leftarrow \mathsf{SiProof}(\mathsf{sk}, m_*, \sigma_*, \mathsf{spk}_*)$ then:

$$\pi_{\mathsf{si},*} \leftarrow \mathsf{V.Proof}(\{\mathsf{spk}_*, \mathsf{pk}_v\}, m_* || \sigma_{1,*}, \sigma_{2,*}, \mathsf{pk}_v, \mathsf{sk}_v)$$

We deduce that the probability that $\mathcal{B}$ wins its experiment is the same as the probability that $\mathcal{A}$ wins its experiments:

$$\mathsf{Pr}[\mathsf{Exp}^{\text{1-non-usu-2}}_{V,\mathcal{B}}(k) = 1] \geq \lambda(k)$$

which conclude the proof. □