SEVIL: Secure and Efficient VerifIcation over Massive Proofs of KnowLedge

Souha Masmoudi[1,2][a], Maryline Laurent[1,2][b] and Nesrine Kaaniche[1,2][c]

[1] *SAMOVAR, Telecom SudParis, Institut Polytechnique de Paris, Evry, France*

[2] *Member of the Chair Values and Policies of Personal Information, Institut Mines-Telecom, Paris, France*

Keywords: Group Signatures, NIWI Proofs, Aggregated Verification, Batch Verification.

Abstract: This paper presents SEVIL, a group signature construction that offers an efficient, aggregated and batch verification over multiple signatures. The proposed group signature scheme is built upon Groth-Sahai Non-Interactive Witness-Indistinguishable proofs, in an effort to reduce the computation complexity, closely associated with the number the number of signatures. SEVIL fulfills the main security and privacy properties, proven through a detailed analysis. The implementation of SEVIL algorithms demonstrates the high efficiency of the aggregated and batch verification with up to 50% of gain in comparison with naive verification of NIWI proofs.

1 INTRODUCTION

The emergence of data-centric applications and services have asserted various concerns related to the massive data collection and analysis by different actors belonging to different levels of trust. Thus, the need to continuously authenticate data origin and verify its integrity has significantly increased. To achieve the aforesaid security requirements, digital signatures are commonly considered as promoting cryptographic primitives and main building blocks of authentication protocols. However, in a multi-owner context, additional requirements are needed, namely *(i)* ensuring that various signers are trustful and *(ii)* protecting their privacy.

For this purpose, group signatures are the best primitive to fulfill the desired properties. In fact, group signatures allow a group member to sign a message on behalf of the group while remaining anonymous. Group signatures support privacy properties, namely unlinkability. Indeed, they prevent both the linkability between multiple signatures issued by the same member and the identification of the signer of a given signature. However, in order to ensure that a signer is trustworthy, the signing key should be verified, which harms the signer's privacy. Thus, to ensure the trade-off between trust and privacy, the group signature scheme might rely on a proof of knowledge (*PoK*) scheme. A proof of knowledge enables a prover (i.e., signer) to convince a verifier that he owns a secret (i.e., a valid pair of keys) without disclosing it, in an interactive session.

Giving consideration to the huge number of proofs collected, in many applications, and to the processing time needed to verify a single proof, there is a crucial need to optimize the verification algorithm by verifying multiple proofs at once. To this question, batch verification has been introduced by Naccache *et al.* (Naccache et al., 1994) enabling the verification of multiple signatures generated by different signers. Batch verification has been applied to some group signature schemes to efficiently verify the huge number of transactions on Blockchain (Zhang et al., 2021) and data transmitted in the Internet of Things (Alamer, 2020). However, to the best of our knowledge, no batch verifier has been constructed over *PoK*-based group signature schemes, i.e., signatures that endorse the verification of the signer key.

In this paper, we present SEVIL, a novel group signature that offers an efficient, aggregated and batch verification over multiple Groth-Sahai Non-Interactive Witness-Indistinguishable proofs (Groth and Sahai, 2008). SEVIL supports a decentralized data certification and a centralized multi-signer data aggregated and batch verification. Indeed, SEVIL allows a signer, belonging to a trusted group, to generate anonymous group signature (i.e., NIWI proof). Relying on this proof, the signer is able to prove to a verifier the integrity of the data without revealing the signing keys. For efficiency reasons, the verifier

[a] https://orcid.org/0000-0002-7194-8240
[b] https://orcid.org/0000-0002-7256-3721
[c] https://orcid.org/0000-0002-1045-6445

Masmoudi, S., Laurent, M. and Kaaniche, N.
SEVIL: Secure and Efficient VerifIcation over Massive Proofs of KnowLedge.
DOI: 10.5220/0011125800003283
In *Proceedings of the 19th International Conference on Security and Cryptography (SECRYPT 2022)*, pages 13-24
ISBN: 978-989-758-590-6; ISSN: 2184-7711

is given the capacity to proceed to the verification of multiple proofs at once. If the batch verification fails, the verifier proceeds to a *divide and conquer* verification. It splits the list of proofs into sub-lists and performs verification to each sub-list recursively until all invalid signatures are identified.

In a nutshell, SEVIL satisfies several properties of interest. First, it ensures data integrity without revealing the data contents, thanks to the use of proofs of knowledge. Second, SEVIL ensures a high level of security as it allows the verifier to check the validity of signers' keys. Third, the proposed system preserves signers' privacy. It ensures that a verifier is not able to link two or several pieces of information signed by the same signer. Fourth, SEVIL proposes a concrete construction of its algorithms, while giving a detailed aggregated and batch verification over Groth-Sahai NIWI proofs. Finally, a complete implemented prototype of SEVIL, introducing two steps of computation improvements (multithreading and preprocessing), proves its efficiency as the aggregated verification reaches a gain of up to 50% compared to the individual verification.

The remainder of this paper is organized as follows. Section 2 presents the design system and goals of this work including the involved entities and the desired security, privacy and performance properties. Section 3 gives an overview of the solution and a high level presentation of its phases and algorithms. After presenting the underlying cryptographic background in Section 4, the proposed scheme is introduced in Section 5. Security and performance discussions are given in Section 6 and Section 7, respectively. Section 8 compares most closely-related works to SEVIL. Finally, Section 9 concludes the paper.

2 DESIGN SYSTEM AND GOALS

This section defines the involved entities and illustrate the design goals that need to be supported by SEVIL.

2.1 Design System

As depicted in Figure 1, SEVIL involves four different entities, defined as follows:

- The trusted authority ($\mathcal{TA}$), is the central entity which is responsible for initializing the whole system.
- The signer ($\mathcal{S}$) signs data on behalf of a group of signers. The group of $\mathcal{S}$s is dynamically generated and managed by a group manager ($\mathcal{G}$).
- The group manager ($\mathcal{G}$) sets-up the group of signers and certifies their keys.
- The verifier ($\mathcal{V}$) checks the correctness of the received data and their associated signatures of multiple signers in a single transaction.

2.2 Design Goals

The design of SEVIL is motivated by the fulfillment of the following security and performance properties.

2.2.1 Security and Privacy Requirements

The proposed SEVIL system aims at ensuring the following security and privacy requirements:

- **unforgeability:** ensures that malicious entities are not able to forge signatures over data.
- **unlinkability:** ensures that an attacker is not able to link several signatures to the same signer.

2.2.2 Performance Requirements

SEVIL system should consider the following requirements for efficiency purposes:

- **Computation Overhead:** SEVIL should provide low computation overhead, i.e. should have efficient verification time even with high number of messages.
- **Communication Overhead:** SEVIL should provide low communication overhead, i.e. should maintain an acceptable communication cost between entities.

3 SEVIL Overview

SEVIL involves three main phases: SETUP, SIGNING and VERIFYING presented hereafter.

The SETUP phase consists of initializing the whole system. It relies on three algorithms, referred to as $\mathsf{Set\_params}$, $\mathsf{Setup\_SGr}_{\mathcal{G}}$ and $\mathsf{Join\_SGr}_{\mathcal{S}/\mathcal{G}}$. During the SETUP phase, a trusted authority generates the system public parameters published to all involved entities, relying on $\mathsf{Set\_params}$ algorithm. The group manager defines the group of signers. It generates the group signature parameters using the $\mathsf{Setup\_SGr}_{\mathcal{G}}$ algorithm and it interacts with each group member (i.e., signer) to derive the associated keys relying on the $\mathsf{Join\_SGr}_{\mathcal{S}/\mathcal{G}}$ algorithm.

The SIGNING phase occurs to sign a message m. Indeed, a signer generates a group signature over the message m, while executing the $\mathsf{G\_Sign}_{\mathcal{S}}$ algorithm. The resulting signature is locally stored.

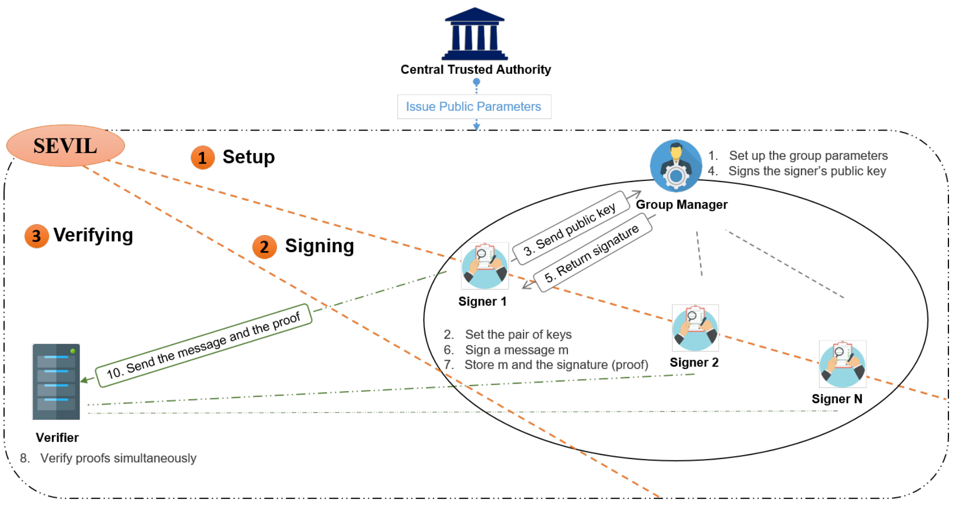
Central Trusted Authority
Issue Public Parameters
SEVIL
1 Setup
2 Signing
3 Verifying
1. Set up the group parameters
4. Signs the signer's public key
Group Manager
3. Send public key
5. Return signature
Signer 1
2. Set the pair of keys
6. Sign a message m
7. Store m and the signature (proof)
Signer 2
Signer N
10. Send the message and the proof
Verifier
8. Verify proofs simultaneously

Figure 1: SEVIL Architecture.

The VERIFYING phase is run by the verifier to check the correctness of the signed data provided by multiple signers in a single transaction. Indeed, $\mathcal{V}$ executes the $\mathsf{Batch\_Verify}_{\mathcal{V}}$ algorithm to verify multiple group signatures issued by members of the same group, in a single verification. Note that if the batch verification fails, $\mathcal{V}$ should proceed progressively by dividing the data list in sub-lists and then by verifying the invalid sub-list message by message. To verify a single message, $\mathcal{V}$ performs the $\mathsf{Agg\_Verify}_{\mathcal{V}}$.

Hereafter, we detail the three phases of SEVIL including the five algorithms that are represented in a chronological sequence in Figure 2. Note that $\mathsf{Agg\_Verify}_{\mathcal{V}}$ algorithm is not represented in the sequence as it occurs only when the batch verification fails.

For ease of presentation, we consider only two signers in the sequence diagram. Recall that verification can be performed on a large number of messages generated by the same signer or by different signers.

3.1 SETUP Phase

The SETUP phase initializes the whole system, relying on three main algorithms, referred to as: $\mathsf{Set\_params}$, $\mathsf{Setup\_SGr}_{\mathcal{G}}$ and $\mathsf{Join\_SGr}_{\mathcal{S}/\mathcal{G}}$ defined as follows.

- $\mathsf{Set\_params}(\lambda) \to pp$ – performed by the trusted authority. Given the security parameter λ, this algorithm generates pp i.e., the system public parameters that will be given as a default input for all the following algorithms.
- $\mathsf{Setup\_SGr}_{\mathcal{G}}() \to (\mathtt{sk}_{\mathcal{G}}, \mathtt{vk}_{\mathcal{G}})$ – run by the group manager in order to set up the group signature parameters. It returns the signers' group verification key $\mathtt{vk}_{\mathcal{G}}$ encompassing $\mathtt{pk}_{\mathcal{G}}$ the public key of the group manager and Σ_{NIWI} of a Non-Interactive Witness-Indistinguishable (NIWI) proof (Groth and Sahai, 2008) associated with the public key. The $\mathsf{Setup\_SGr}_{\mathcal{G}}$ algorithm also outputs the secret key $\mathtt{sk}_{\mathcal{G}}$ of $\mathcal{G}$.
- $\mathsf{Join\_SGr}_{\mathcal{S}/\mathcal{G}}(\mathtt{sk}_{\mathcal{G}}) \to (\mathtt{sk}_{\mathcal{S}}, \mathtt{pk}_{\mathcal{S}}, \sigma_k)$ – executed through an interactive session between the signer and the group manager. It takes as input the secret key $\mathtt{sk}_{\mathcal{G}}$ of the group manager, and outputs the pair of keys $(\mathtt{sk}_{\mathcal{S}}, \mathtt{pk}_{\mathcal{S}})$ of the group member (i.e., signer) $\mathcal{S}$, and a signature σ_k over $\mathcal{S}$'s public key $\mathtt{pk}_{\mathcal{S}}$. Indeed, $\mathcal{S}$ is responsible for generating his pair of keys, while $\mathcal{G}$ is in charge of the signature σ_k generation aiming to certify $\mathcal{S}$'s keys.

3.2 SIGNING Phase

The SIGNING phase is performed by a signer. In this phase, $\mathcal{S}$ relies on the $\mathsf{G\_Sign}_{\mathcal{S}}$ algorithm to sign a message $m \in \mathbb{G}_2$:

- $\mathsf{G\_Sign}_{\mathcal{S}}(\mathtt{vk}_{\mathcal{G}}, \mathtt{sk}_{\mathcal{S}}, \mathtt{pk}_{\mathcal{S}}, \sigma_k, m) \to (\sigma_m, \Pi)$ – performed by the signer as a group member. This algorithm takes as input the group public parameters $\mathtt{vk}_{\mathcal{G}}$, the pair of keys $(\mathtt{sk}_{\mathcal{S}}, \mathtt{pk}_{\mathcal{S}})$ of $\mathcal{S}$, the signature σ_k over $\mathcal{S}$'s public key and the message m. The $\mathsf{G\_Sign}_{\mathcal{S}}$ returns a signature σ_m over the message m and a NIWI proof Π over the two signatures σ_k and σ_m. The proof Π is locally stored with the message m.

Note that the NIWI proof represents the group signature generated by $\mathcal{S}$ as a group member.

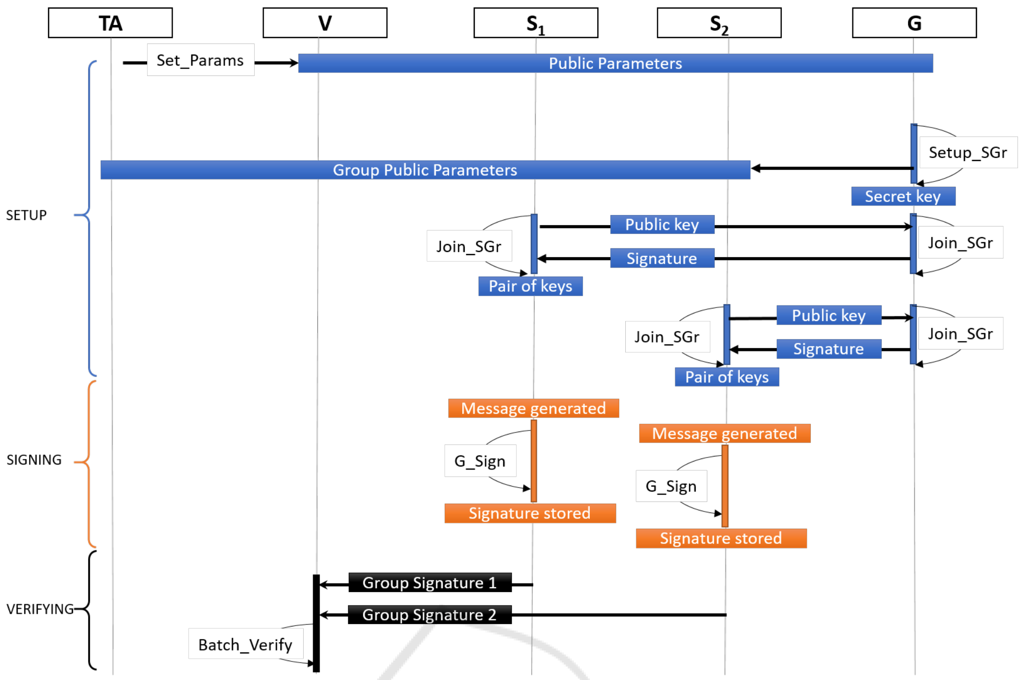

Figure 2: Workflow of the SEVIL system.

3.3 VERIFYING Phase

The VERIFYING phase involves two main algorithms referred to as Batch_Verify$_{\mathcal{V}}$ and Agg_Verify$_{\mathcal{V}}$:

- Batch_Verify$_{\mathcal{V}}(\mathtt{vk}_{\mathcal{G}}, \{m_i, \Pi_i\}_{i=1}^{N}) \rightarrow b$ – performed by $\mathcal{V}$. Given the public parameters $\mathtt{vk}_{\mathcal{G}}$, a list of N messages m_i and N corresponding proofs Π_i sent by multiple signers (i.e., a signer can send to $\mathcal{V}$ more than one message), the Batch_Verify$_{\mathcal{V}}$ algorithm returns a bit $b \in \{0,1\}$ stating whether the list of proofs is valid or not.
- Agg_Verify$_{\mathcal{V}}(\mathtt{vk}_{\mathcal{G}}, m, \Pi) \rightarrow b$ – performed by $\mathcal{V}$ when Batch_Verify$_{\mathcal{V}}$ returns 0 over a list or a sub-list of messages. Given the public parameters $\mathtt{vk}_{\mathcal{G}}$, a message m and the corresponding proof Π, from an invalid sub-list, the Agg_Verify$_{\mathcal{V}}$ algorithm returns a bit $b \in \{0,1\}$ stating whether proof is valid or not.

4 CRYPTOGRAPHIC BACKGROUND

In this section, we first present the Non-Interactive Witness-Indistinguishable (NIWI) proofs introduced by Groth and Sahai (Groth and Sahai, 2008). Then, we propose a group signature scheme built upon NIWI proofs.

4.1 Non-Interactive Witness-Indistinguishable Proofs

Here, we present the Groth-Sahai NIWI proof scheme applied to pairing product equations when considering an asymmetric bilinear map.

The NIWI scheme, involves four PPT algorithms (NIWI.Setup, NIWI.CRS, NIWI.Proof, NIWI.Verify):

- NIWI.Setup: This algorithm outputs a setup $(\mathtt{gk}, \mathtt{sk})$ such that $\mathtt{gk} = (n, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, g_1, g_2, e)$ and $\mathtt{sk} = (p, q)$ where $n = pq$.
- NIWI.CRS: This algorithm generates a common reference string CRS. It takes $(\mathtt{gk}, \mathtt{sk})$ as inputs and produces $\mathtt{CRS} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, e, \iota_1, p_1, \iota_2, p_2, \iota_3, \mathtt{U}, \mathtt{V})$, where $\mathtt{U} = rg_1$, $\mathtt{V} = sg_2$; $r, s \in \mathbb{Z}_n^*$ and

$$\iota_1 : \begin{array}{c} \mathbb{G}_1 \rightarrow \mathbb{G}_1 \\ x \mapsto x \end{array} \quad \iota_2 : \begin{array}{c} \mathbb{G}_2 \rightarrow \mathbb{G}_2 \\ y \mapsto y \end{array} \quad \iota_3 : \begin{array}{c} \mathbb{G}_3 \rightarrow \mathbb{G}_3 \\ z \mapsto z \end{array}$$

$$p_1 : \begin{array}{c} \mathbb{G}_1 \rightarrow \mathbb{G}_1 \\ x \mapsto \lambda x \end{array} \quad p_2 : \begin{array}{c} \mathbb{G}_2 \rightarrow \mathbb{G}_2 \\ y \mapsto \lambda y \end{array} \quad p_3 : \begin{array}{c} \mathbb{G}_3 \rightarrow \mathbb{G}_3 \\ z \mapsto z^{\lambda} \end{array}$$

- NIWI.Proof: This algorithm generates a NIWI proof for satisfiability of a set of pairing product equations of the form of

$$\prod_{i=1}^{k} e(\mathcal{A}_i, \mathcal{Y}_i) \prod_{i=1}^{l} e(\mathcal{X}_i, \mathcal{B}_i) \prod_{i=1}^{l} \prod_{j=1}^{k} e(\mathcal{X}_i, \mathcal{Y}_i)^{\gamma_{ij}} = t$$

also written as

$$(\vec{\mathcal{A}} \cdot \vec{\mathcal{Y}})(\vec{\mathcal{X}} \cdot \vec{\mathcal{B}})(\vec{\mathcal{X}} \cdot \Gamma\vec{\mathcal{Y}}) = t$$

It takes as input gk, CRS and a list of pairing product equations $\{(\vec{\mathcal{A}}_i, \vec{\mathcal{B}}_i, \Gamma_i, t_i)\}_{i=1}^N$ and a satisfying witness $\vec{\mathcal{X}} \in \mathbb{G}_1^k$, $\vec{\mathcal{Y}} \in \mathbb{G}_2^l$. To generate a proof over a pairing product equation, the algorithm, first, picks at random $\mathcal{R} \leftarrow Vec_k(\mathbb{Z}_n)$ and $\mathcal{S} \leftarrow Vec_l(\mathbb{Z}_n)$, commits to all variables as $\vec{\mathcal{C}} := \vec{\mathcal{X}} + \mathcal{R}\mathtt{U}$ and $\vec{\mathcal{D}} := \vec{\mathcal{Y}} + \mathcal{S}\mathtt{V}$, and computes

$$\pi = \mathcal{R}^\top \iota_2(\vec{\mathcal{B}}) + \mathcal{R}^\top \Gamma \iota_2(\vec{\mathcal{Y}}) + \mathcal{R}^\top \Gamma \mathcal{S} \mathtt{V}$$
$$\theta = \mathcal{S}^\top \iota_1(\vec{\mathcal{A}}) + \mathcal{S}^\top \Gamma^\top \iota_1(\vec{\mathcal{X}})$$

The algorithm outputs the proof (π, θ).

- NIWI.Verify: This algorithm checks if the proof is valid. It takes gk, CRS, $\{(\vec{\mathcal{A}}_i, \vec{\mathcal{B}}_i, \Gamma_i, t_i)\}_{i=1}^N$ and $\{(\vec{\mathcal{C}}_i, \vec{\mathcal{D}}_i, \pi_i, \theta_i)\}_{i=1}^N$ as inputs and for each equation, checks the following equation:

$$e(\iota_1(\vec{\mathcal{A}}_i), \vec{\mathcal{D}}_i) e(\vec{\mathcal{C}}_i, \iota_2(\vec{\mathcal{B}}_i)) e(\vec{\mathcal{C}}_i, \Gamma_i \vec{\mathcal{D}}_i) = \iota_3(t_i) e(\mathtt{U}, \pi_i) e(\theta_i, \mathtt{V}) \quad (1)$$

The algorithm outputs 1 if the equation holds, else it outputs 0.

4.2 Group Signatures

We present an instantiation of a group signature scheme that relies on a witness-indistinguishable proof of knowledge system NIWI and structure-preserving signatures.

A group signature scheme GSIG relies on the four following algorithms (GSIG.Setup, GSIG.Join, GSIG.Sign, GSIG.Verify):

- GSIG.Setup : represents the setup algorithm. It generates the key pair $(\mathtt{sk_g}, \mathtt{pk}_g)$ of the group manager and sets up a CRS Σ_{NIWI} for the NIWI proof. The group verification key is set as $\mathtt{vk}_g = (\mathtt{pk}_g, \Sigma_{\mathsf{NIWI}})$, while the certification secret key $\mathtt{sk_g}$ is privately stored by the group manager.
- GSIG.Join: represents the join algorithm. It is composed of two steps. In the first one, the group member generates his own key-pair $(\mathtt{sk_p}, \mathtt{pk}_p)$. The public key $\mathtt{pk}_p$ is sent to the group manager. This latter generates a signature σ_p over the key $\mathtt{pk}_p$ that he sends to the group member.
- GSIG.Sign: represents the signing algorithm run by a group member on a message $m \in \mathbb{Z}_q$. The group member generates, over the message m, a signature σ_m and a NIWI proof Π.
- GSIG.Verify: represents the group signature verification algorithm run by a verifier. It takes $(\mathtt{vk}_g, m, \Pi)$ as input and verifies the correctness of the NIWI proof Π w.r.t. $pub = (\mathtt{pk}_g, m)$ and the CRS Σ_{NIWI}.

5 SEVIL ALGORITHMS

This section gives the concrete construction of SEVIL. Based on the Groth-Sahai NIWI proof scheme proposed in Section 4, it details the three phases of the SEVIL, including the six algorithms presented in Section 3 .

5.1 SETUP Phase

- Set_params – given λ, $\mathcal{TA}$ selects an asymmetric bilinear group $(n, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, g_1, g_2, e)$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic groups of prime order n, g_1 and g_2 are generators of respectively $\mathbb{G}_1$ and $\mathbb{G}_2$ and e is a bilinear map such that $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$. The Set_params algorithm outputs the tuple $(n, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, g_1, g_2, e)$ denoted by *pp*. *pp* represents the system global parameters that are given as a default input to all the algorithms run by the system's entities.
- Setup_SGr$_\mathcal{G}$ – $\mathcal{G}$ sets up the public parameters of the group of signers. It generates a group public key $\mathtt{vk}_\mathcal{G}$ and a certification secret key $\mathtt{sk}_\mathcal{G}$ as detailed in Algorithm 1.
- Join_SGr$_{\mathcal{S}/\mathcal{G}}$ – First, $\mathcal{S}$ generates his pair of keys $(\mathtt{sk}_\mathcal{S}, \mathtt{pk}_\mathcal{S})$ as depicted in Algorithm 2 (line 4 – line 9). Afterwards, $\mathcal{G}$ generates a signature σ_k over the public key $\mathtt{pk}_\mathcal{S}$ as detailed in Algorithm 2 (lines 11 – 17).

5.2 SIGNING Phase

- G_Sign$_\mathcal{S}$ – $\mathcal{S}$ sets a message $m \in \mathbb{G}_2$ $\mathcal{S}$ that he signs on behalf of the group. Indeed, $\mathcal{S}$ generates a signature σ_m over the message m, and computes a proof Π over the signatures σ_k and σ_m as shown in Algorithm 3.

5.3 VERIFYING Phase

- Batch_Verify$_\mathcal{V}$ – We consider a list of N messages m_i and the corresponding proofs Π_i. Each proof Π_i is composed of six sub-proofs (i.e., two sub-proofs generated over the signature σ_m w.r.t. the message m, and four sub-proofs generated over the signature σ_k w.r.t. the signer key $\mathtt{pk}_\mathcal{S}$). The list of proofs can be presented as follows:

$$\begin{cases} \{(\vec{\mathcal{A}}_{ijm}, \vec{\mathcal{B}}_{ijm}, \Gamma_{ijm}, t_{ijm})\}_{i,j=1}^{i=N,j=2}, \\ \{(\vec{\mathcal{C}}_{ijm}, \vec{\mathcal{D}}_{ijm}, \pi_{ijm}, \theta_{ijm})\}_{i,j=1}^{i=N,j=2}, \\ \{(\vec{\mathcal{A}}_{ilk}, \vec{\mathcal{B}}_{ilk}, \Gamma_{ilk}, t_{ilk})\}_{i,l=1}^{i=N,l=4}, \\ \{(\vec{\mathcal{C}}_{ilk}, \vec{\mathcal{D}}_{ilk}, \pi_{ilk}, \theta_{ilk})\}_{i,l=1}^{i=N,l=4}. \end{cases}$$

```
1: Input: the system public parameters $pp$
2: Output: the group public parameters $\mathtt{vk}_{\mathcal{G}}$ and the secret key $\mathtt{sk}_{\mathcal{G}}$
3: // The next iterations are executed to generate the pair of keys of $\mathcal{G}$
4: pick at random $g_{r1}, h_{u1} \leftarrow \mathbb{G}_1^*$, $g_{r2}, h_{u2} \leftarrow \mathbb{G}_2^*$ for $i = 1$ to 2 do
       pick at random $\gamma_{1i}, \delta_{1i} \leftarrow \mathbb{Z}_n^*$ compute $g_{1i} \leftarrow {g_{r1}}^{\gamma_{1i}}$, $h_{1i} \leftarrow {h_{u1}}^{\delta_{1i}}$
   end
   for $j = 1$ to 7 do
       pick at random $\gamma_{2j}, \delta_{2j} \leftarrow \mathbb{Z}_n^*$ compute $g_{2i} \leftarrow {g_{r2}}^{\gamma_{2j}}$ and $h_{2j} \leftarrow {h_{u2}}^{\delta_{2j}}$
   end
5: pick at random $\gamma_{1z}, \delta_{1z}, \gamma_{2z}, \delta_{2z} \leftarrow \mathbb{Z}_n^*$ ;
6: compute $g_{1z} \leftarrow {g_{r1}}^{\gamma_{1z}}$, $h_{1z} \leftarrow {h_{u1}}^{\delta_{1z}}$, $g_{2z} \leftarrow {g_{r2}}^{\gamma_{2z}}$ and $h_{2z} \leftarrow {h_{u2}}^{\delta_{2z}}$ ;
7: pick at random $\alpha_1, \alpha_2, \beta_1, \beta_2 \leftarrow \mathbb{Z}_n^*$ ;
8: $\mathtt{pk}_1 \leftarrow (g_{2z}, h_{2z}, g_{2r}, h_{2u}, g_1^{\alpha_2}, g_1^{\beta_2}, \{g_{2j}, h_{2j}\}_{j=1}^7)$ and $\mathtt{sk}_1 \leftarrow (\mathtt{pk}_1, \alpha_2, \beta_2, \gamma_{2z}, \delta_{2z}, \{\gamma_{2j}, \delta_{2j}\}_{j=1}^7)$ ;
9: $\mathtt{pk}_2 \leftarrow (g_{1z}, h_{1z}, g_{1r}, h_{1u}, g_2^{\alpha_1}, g_2^{\beta_1}, \{g_{1i}, h_{1i}\}_{i=1}^2)$ and $\mathtt{sk}_2 \leftarrow (\mathtt{pk}_2, \alpha_1, \beta_1, \gamma_{1z}, \delta_{1z}, \{\gamma_{1i}, \delta_{1i}\}_{i=1}^2)$ ;
10: set $\mathtt{pk}_g \leftarrow (\mathtt{pk}_1, \mathtt{pk}_2)$ and $\mathtt{sk}_g \leftarrow (\mathtt{sk}_1, \mathtt{sk}_2)$ ;
11: // The next iterations are executed to generate $\Sigma_{\mathsf{NIWI}}$
12: pick at random $r, s \leftarrow \mathbb{Z}_n^*$ and set $\mathtt{U} = rg_1$ and $\mathtt{V} = sg_2$ ;
13: set $\Sigma_{\mathsf{NIWI}} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, e, \iota_1, p_1, \iota_2, p_2, \iota_3, \mathtt{U}, \mathtt{V})$ ;
14: $\mathtt{vk}_{\mathcal{G}} \leftarrow (\mathtt{pk}_g, \Sigma_{\mathsf{NIWI}})$ ;
15: return $(\mathtt{sk}_{\mathcal{G}}, \mathtt{vk}_{\mathcal{G}})$
```

Algorithm 1: $\mathsf{Setup\_SGr}_{\mathcal{G}}$ algorithm.

Referring to the generation of the group signature, the tuples $\{(\vec{\mathcal{A}}_{jm}, \vec{\mathcal{B}}_{jm}, \Gamma_{jm}, t_{jm})\}_{j=1}^2$ and $\{(\vec{\mathcal{A}}_{lk}, \vec{\mathcal{B}}_{lk}, \Gamma_{lk}, t_{lk})\}_{l=1}^4$ are unchanged for all N proofs and all signers. Thus, for a given list of messages, $\mathcal{V}$ verifies the validity of the proofs by checking if equations (2) and (3), depicted in Figure 3, hold:

- $\mathsf{Agg\_Verify}_{\mathcal{V}}$ – We consider a message m belonging to an invalid proof-list and its corresponding proof Π. Using the tuples $\{(\vec{\mathcal{A}}_{jm}, \vec{\mathcal{B}}_{jm}, \Gamma_{jm}, t_{jm})\}_{j=1}^2$ and $\{(\vec{\mathcal{A}}_{lk}, \vec{\mathcal{B}}_{lk}, \Gamma_{lk}, t_{lk})\}_{l=1}^4$ along with the tuples $\{(\vec{\mathcal{C}}_{jm}, \vec{\mathcal{D}}_{jm}, \pi_{jm}, \theta_{jm})\}_{j=1}^{j=2}$ and $\{(\vec{\mathcal{C}}_{lk}, \vec{\mathcal{D}}_{lk}, \pi_{lk}, \theta_{lk})\}_{l=1}^{l=4}$ derived from Π, $\mathcal{V}$ checks if equations (4) and (5), depicted in Figure 4, hold.

```
1: Input: the security parameter $\lambda$ and the secret key of the group manager $\mathtt{sk}_{\mathcal{G}}$
2: Output: the pair of keys of a signer $(\mathtt{sk}_{\mathcal{S}}, \mathtt{pk}_{\mathcal{S}})$ and the signature $\sigma_k$ over the public key $\mathtt{pk}_{\mathcal{S}}$
3: // The next is set by $\mathcal{S}$
4: pick at random $g_r, h_u \leftarrow \mathbb{G}_1^*$, $\gamma, \delta \leftarrow \mathbb{Z}_n^*$ ;
5: compute $g_\gamma \leftarrow {g_r}^{\gamma}$ and $h_\delta \leftarrow {h_u}^{\delta}$ ;
6: pick at random $\gamma_z, \delta_z \leftarrow \mathbb{Z}_n^*$ ;
7: compute $g_z \leftarrow {g_r}^{\gamma_z}$ and $h_z \leftarrow {h_u}^{\delta_z}$ ;
8: pick at random $\alpha, \beta \leftarrow \mathbb{Z}_n^*$ ;
9: set $\mathtt{pk}_{\mathcal{S}} = (g_z, h_z, g_r, h_u, g_2^{\alpha}, g_2^{\beta}, g_\gamma, h_\delta)$ and $\mathtt{sk}_{\mathcal{S}} = (\mathtt{pk}_{\mathcal{S}}, \alpha, \beta, \gamma_z, \delta_z, \gamma, \delta)$ ;
10: // The next is set by $\mathcal{G}$
11: pick at random $\zeta_2, \rho_2, \tau_2, \varphi_2, \omega_2 \leftarrow \mathbb{Z}_n^*$ ;
12: compute $z_2 = g_2^{\zeta_2}$, $s_2 = {g_{1r}}^{\rho_2}$, $t_2 = {g_2}^{\tau_2}$, $r_2 = {g_2}^{\alpha_1 - \rho_2\tau_2 - \gamma_{1z}\zeta_2} \prod_{i=1}^2 {m_{2i}}^{-\gamma_{1i}}$, $u_2 = {g_2}^{\beta_1 - \varphi_2\omega_2 - \delta_{1z}\zeta_2} \prod_{i=1}^2 {m_{2i}}^{-\delta_{1i}}$, $v_2 = {h_{1u}}^{\varphi_2}$, $w_2 = {g_2}^{\omega_2}$ ; where $\vec{m}_2 = (g_2^{\alpha}, g_2^{\beta})$ ;
13: set $\sigma_2 = (z_2, r_2, s_2, t_2, u_2, v_2, w_2)$ ;
14: pick at random $\zeta_1, \rho_1, \tau_1, \varphi_1, \omega_1 \leftarrow \mathbb{Z}_n^*$ ;
15: compute $z_1 = g_1^{\zeta_1}$, $s_1 = {g_{2r}}^{\rho_1}$, $t_1 = {g_1}^{\tau_1}$, $r_1 = {g_1}^{\alpha_2 - \rho_1\tau_1 - \gamma_{2z}\zeta_1} \prod_{j=1}^7 {m_{1j}}^{-\gamma_{2j}}$, $u_1 = {g_1}^{\beta_2 - \varphi_1\omega_1 - \delta_{2z}\zeta_1} \prod_{j=1}^7 {m_{1j}}^{-\delta_{2j}}$, $v_1 = {h_{2u}}^{\varphi_1}$, $w_1 = {g_1}^{\omega_1}$; where $\vec{m}_1 = (g_z, h_z, g_r, h_u, g_\gamma, h_\delta, s_2)$;
16: set $\sigma_1 = (z_1, r_1, s_1, t_1, u_1, v_1, w_1)$ ;
17: set $\sigma_k = (\sigma_1, \sigma_2)$ ;
18: return $(\mathtt{sk}_{\mathcal{S}}, \mathtt{pk}_{\mathcal{S}}, \sigma_k)$
```

Algorithm 2: $\mathsf{Join\_SGr}_{\mathcal{S}/\mathcal{G}}$ algorithm.

6 SECURITY DISCUSSION

In this section, we prove that SEVIL's construction is secure and privacy-preserving w.r.t. the properties defined in Section 2.2. In order to appropriately address security and privacy requirements mentioned is Section 2.2.1, we consider two main adversaries, as follows:

- **A Malicious Adversary:** attempts, by himself or when colluding with other malicious adversaries, to generate a valid group signature over a message without being authorized by the group manager. A malicious adversary plays the role of an outsider and is mainly considered against unforgeability property.
- **A Honest but Curious Adversary:** given a valid group signature, a honest but curious adversary tries to identify the signer of a particular message. He may also attempt to link two signatures issued by the same group member. A curious adversary plays the role of a verifier ($\mathcal{V}$), and considered against unlinkability[1] requirement.

[1]We assume that unlinkability property includes the anonymity of signers as group members.

$$\prod_i\prod_j \left(e(\vec{\mathcal{C}}_{ijm}, \Gamma_m \vec{\mathcal{D}}_{ijm})\right) = e(\mathtt{U}, \sum_i\sum_j \pi_{ijm}) e(\sum_i\sum_j \theta_{ijm}, \mathtt{V}) \quad (2)$$

$$\prod_l e(\iota_1(\vec{\mathcal{A}}_{lk}), \sum_i \vec{\mathcal{D}}_{ilk}) e(\sum_i \vec{\mathcal{C}}_{ilk}, \iota_2(\vec{\mathcal{B}}_{lk})) \prod_i\prod_l \left(e(\vec{\mathcal{C}}_{ilk}, \Gamma_{lk} \vec{\mathcal{D}}_{ilk})\right) = \left(\prod_l \iota_3(t_l k)^N\right) e(\mathtt{U}, \sum_i\sum_l \pi_{ilk}) e(\sum_i\sum_l \theta_{ilk}, \mathtt{V}) \quad (3)$$

Figure 3: Verification equations of $\mathsf{Batch\_Verify}_{\mathcal{V}}$ algorithm.

$$\prod_j \left(e(\vec{\mathcal{C}}_{jm}, \Gamma_m \vec{\mathcal{D}}_{jm})\right) = e(\mathtt{U}, \sum_j \pi_{jm}) e(\sum_j \theta_{jm}, \mathtt{V}) \quad (4)$$

$$\prod_l e(\iota_1(\vec{\mathcal{A}}_{lk}), \vec{\mathcal{D}}_{lk}) e(\vec{\mathcal{C}}_{lk}, \iota_2(\vec{\mathcal{B}}_{lk})) \prod_l \left(e(\vec{\mathcal{C}}_{lk}, \Gamma_{lk} \vec{\mathcal{D}}_{lk})\right) = \left(\prod_l \iota_3(t_l k)\right) e(\mathtt{U}, \sum_l \pi_{lk}) e(\sum_l \theta_{lk}, \mathtt{V}) \quad (5)$$

Figure 4: Verification equations of $\mathsf{Agg\_Verify}_{\mathcal{V}}$ algorithm.

```
1:  Input: the public parameters of the signers' group $\mathtt{vk}_{\mathcal{G}}$, the pair of keys $(\mathtt{sk}_{\mathcal{S}}, \mathtt{pk}_{\mathcal{S}})$, the signature $\sigma_k$ over the
    signer public key and the message $m$
2:  Output: a signature $\sigma_m$ over the message $m$ and a proof $\Pi$ over the signatures $\sigma_k$ and $\sigma_m$
3:  // The next is executed by $\mathcal{S}$ to sign $m$
4:  pick at random $\zeta, \rho, \tau, \varphi, \omega \leftarrow \mathbb{Z}_n^*$ ;
5:  run $z = g_2^{\zeta}$, $r = {g_2}^{\alpha-\rho\tau-\gamma_z\zeta} m^{-\gamma}$, $s = {g_r}^{\rho}$, $t = {g_2}^{\tau}$, $u = {g_2}^{\beta-\varphi\omega-\delta_z\zeta} m^{-\delta}$, $v = {h_u}^{\varphi}$, $w = {g_2}^{\omega}$ ;
6:  set $\sigma_m = (z, r, s, t, u, v, w)$ ;
7:  // The next is set to generate a proof on equations $\{(\vec{\mathcal{A}}_{im}, \vec{\mathcal{B}}_{im}, \Gamma_{im}, t_{im})\}_{i=1}^{2}$ where $\vec{\mathcal{A}}_{im} = \vec{\mathcal{B}}_{im} = \vec{0}$,
    $\Gamma_{im} = \mathcal{MAT}_{3\times3}(1)$ for $i = 1,2$, $t_{1m} = t_{2m} = 1_{\mathbb{G}_3}$
8:  $\vec{\mathcal{X}}_{1m} = (g_z, g_r, s)$, $\vec{\mathcal{X}}_{2m} = (h_z, h_u, v)$ , $\vec{\mathcal{Y}}_{1m} = (z, {g_2}^{\alpha-\rho\tau-\gamma_z\zeta}, t)$ and $\vec{\mathcal{Y}}_{2m} = (z, {g_2}^{\beta-\rho\tau-\delta z\zeta}, w)$ ;
9:  $\pi_m = \{(\vec{\mathcal{C}}_{im}, \vec{\mathcal{D}}_{im}, \pi_{im}, \theta_{im})\}_{i=1}^{2} \leftarrow \mathsf{NIWI.Proof}(\mathtt{vk}_g, \{(\vec{\mathcal{A}}_{im}, \vec{\mathcal{B}}_{im}, \Gamma_{im}, t_{im})\}_{i=1}^{2}, \{(\vec{\mathcal{X}}_{im}, \vec{\mathcal{Y}}_{im})\}_{i=1}^{2})$ ;
10: // The next is set to generate a proof on equations $\{(\vec{\mathcal{A}}_{ik}, \vec{\mathcal{B}}_{ik}, \Gamma_{ik}, t_{ik})\}_{i=1}^{4}$ where $\vec{\mathcal{A}}_{1k} = (g_1^{\alpha_2})$, $\vec{\mathcal{A}}_{2k} = (g_1^{\beta_2})$, $\vec{\mathcal{A}}_{3k} =$
    $(g_{1z}, g_{1r})$, $\vec{\mathcal{A}}_{4k} = (h_{1z}, h_{1u})$, $\vec{\mathcal{B}}_{1k} = (g_{2z}, g_{2r})$, $\vec{\mathcal{B}}_{2k} = (h_{2z}, h_{2u})$, $\vec{\mathcal{B}}_{3k} = (g_2^{\alpha_1})$, $\vec{\mathcal{B}}_{4k} = (g_2^{\beta_1})$, $\Gamma_{1k} = (\gamma_{2z}, -1)$, $\Gamma_{2k} = (\delta_{2z}, -1)$,
    $\Gamma_{3k} = (\gamma_{1z}, -1)$, $\Gamma_{4k} = (\delta_{1z}, -1)$, $t_{1k} = e(g_1^{\alpha_2}, g_{2r})$, $t_{2k} = e(g_1^{\beta_2}, h_{2u})$, $t_{3k} = e(g_{1r}, g_2^{\alpha_1})$ and $t_{4k} = e(h_{1u}, g_2^{\beta_1})$
11: $\vec{\mathcal{X}}_{1k} = (z_1, {g_1}^{\alpha_2-\rho_1\tau_1-\gamma_{2z}\zeta_1})$, $\vec{\mathcal{X}}_{2k} = (z_1, {g_1}^{\beta_2-\rho_1\tau_1-\delta_{2z}\zeta_1})$, $\vec{\mathcal{X}}_{3k} = (g_{1r})$, $\vec{\mathcal{X}}_{4k} = (h_{1u})$ , $\vec{\mathcal{Y}}_{1k} = (g_{2r})$, $\vec{\mathcal{Y}}_{2k} = (h_{2u})$, $\vec{\mathcal{Y}}_{3k} =$
    $(z_2, {g_2}^{\alpha_1-\rho_2\tau_2-\gamma_{1z}\zeta_2})$, and $\vec{\mathcal{Y}}_{4k} = (z_2, {g_2}^{\beta_1-\rho_2\tau_2-\delta_{1z}\zeta_2})$ ;
12: $\pi_p = \{(\vec{\mathcal{C}}_{ik}, \vec{\mathcal{D}}_{ik}, \pi_{ik}, \theta_{ik})\}_{i=1}^{4} \leftarrow \mathsf{NIWI.Proof}(\mathtt{vk}_g, \{(\vec{\mathcal{A}}_{ik}, \vec{\mathcal{B}}_{ik}, \Gamma_{ik}, t_{ik})\}_{i=1}^{4}, \{(\vec{\mathcal{X}}_{ik}, \vec{\mathcal{Y}}_{ik})\}_{i=1}^{4})$ ;
13: set $\pi_k = ((\pi_{ik}, \theta_{ik})_{i=1}^{4})$ ;
14: set $\Pi = (\pi_k, \pi_m)$ ;
15: return $(\sigma_m, \Pi)$
```

Algorithm 3: $\mathsf{G\_Sign}_{\mathcal{S}}$ algorithm.

6.1 Unforgeability

The unforgeability property states that a malicious outsider is not able to forge the SEVIL group signature.

Let us consider an adversary $\mathcal{A}$ that is allowed to query, as many times as he wants, the $\mathsf{G\_Sign}$ algorithm on a message m_i. Then, during the challenge phase, $\mathcal{A}$ is asked to produce a valid message signature pair (m^*, Π^*) such that the message m^* was not queried before.

For this purpose, we consider two adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ respectively against the unforgeability of the group signature[2] scheme GSIG and the soundness[3] of the proof system NIWI. The advantage of $\mathcal{A}$ to break SEVIL unforgeability is expressed as follows:

$$\mathbf{Adv}_{\mathcal{A}}^{unfor}(1^{\lambda}) \leq \mathbf{Adv}_{\mathsf{GSIG},\mathcal{B}_1}^{unfor}(1^{\lambda}) + \mathbf{Adv}_{\mathsf{NIWI},\mathcal{B}_2}^{sound}(1^{\lambda})$$

According to (Bellare et al., 2003), the unforgeability property can be directly inherited from the traceability property stating that it is not possible to generate signatures without tracing its originator. As the group signature scheme GSIG, relying on the construction of Bellare *et al.* (Bellare et al., 2003), is proven to be traceable, then the unforgeability property of GSIG is also satisfied. Thus, the $\mathbf{Adv}_{\mathsf{GSIG},\mathcal{B}_1}^{unfor}(1^{\lambda})$ function is negligible. The advantage function $\mathbf{Adv}_{\mathsf{NIWI},\mathcal{B}_2}^{sound}(1^{\lambda})$ can be expressed as follows:

$$\mathbf{Adv}_{\mathsf{NIWI},\mathcal{B}_2}^{sound}(1^{\lambda}) = Pr[\mathcal{B}_2 \quad outputs \quad (m,\Pi) : \mathsf{NIWI.Verify}(m,\Pi) = 1]$$

Referring to (Bellare et al., 2003), $Pr[\mathcal{B}_3 \quad outputs \quad (m,\Pi) : \mathsf{NIWI.Verify}(m,\Pi) = 1] \leq 2^{-\lambda}$ as the NIWI proof system is sound. As

[2]Unforgeability of group signatures states that it is not possible to generate a valid message signature pair unless secret keys are known.

[3]The soundness property ensures that is it not possible to prove a false statement.

such, the advantage function $\mathbf{Adv}^{sound}_{\mathsf{NIWI},\mathcal{B}_3}(1^\lambda)$ is negligible. Thus, the advantage function $\mathbf{Adv}^{unfor}_{\mathcal{A}}(1^\lambda)$ is also negligible proving that SEVIL satisfies the unforgeability property.

6.2 Unlinkability

The unlinkability property states that a curious verifier is not able neither to link two or several group signatures issued by the same signer nor to identify the originator of a group signature.

Let us consider an adversary $\mathcal{A}$ that is allowed to query, as many times as he wants, the G_Sign algorithm on the same message m^*, for two signers $\mathcal{S}_0$ and $\mathcal{S}_1$. For each session, $\mathcal{A}$ receives a group signature represented by the tuple $((\vec{\mathcal{C}}^i_{jm}, \vec{\mathcal{D}}^i_{jm}, \pi^i_{jm}, \theta^i_{jm})^2_{j=1}, (\vec{\mathcal{C}}^i_{jk}, \vec{\mathcal{D}}^i_{jk}, \pi^i_{jk}, \theta^i_{jk})^4_{j=1})$. Afterwards, $\mathcal{A}$ is given a pair of group signatures over the same message m^*. The first signature is generated for the signer $\mathcal{S}_0$ and is represented by the tuple $((\vec{\mathcal{C}}^*_{jm}, \vec{\mathcal{D}}^*_{jm}, \pi^*_{jm}, \theta^*_{jm})^2_{j=1}, (\vec{\mathcal{C}}^*_{jk}, \vec{\mathcal{D}}^*_{jk}, \pi^*_{jk}, \theta^*_{jk})^4_{j=1})$. The second signature is generated either for signer $\mathcal{S}_0$ or signer $\mathcal{S}_1$, according to a randomly selected bit $b \in \{0,1\}$. This second signature is represented by the tuple $((\vec{\mathcal{C}}^b_{jm}, \vec{\mathcal{D}}^b_{jm}, \pi^b_{jm}, \theta^b_{jm})^2_{j=1}, (\vec{\mathcal{C}}^b_{jk}, \vec{\mathcal{D}}^b_{jk}, \pi^b_{jk}, \theta^b_{jk})^4_{j=1})$. $\mathcal{A}$ is asked to guess if the two group signatures are generated by the same signer or two different signers with a probability greater than $\frac{1}{2}$.

Let us suppose that $\mathcal{A}$ has an advantage ε against the unlinkability property of SEVIL. A simulator $\mathcal{B}$ against the computational witness-indistinguishability property can be constructed with the help of the adversary $\mathcal{A}$. Indeed, $\mathcal{B}$ is given two commitments (C,D) and (C_b,D_b). The commitment (C,D) is generated over a witness (X_0,Y_0), while (C_b,D_b) is computed over a witness (X_b,Y_b), where $b \in \{0,1\}$. $\mathcal{B}$ is asked, by its own challenger $\mathcal{C}$, to guess guess the bit b i.e., guess whether the commitments were generated over the same witness or two different witnesses. Thus, $\mathcal{B}$ selects two tuples (A_0,B_0,Γ_0,t_0) and (A_b,B_b,Γ_b,t_b) and computes the corresponding proofs (π_0,θ_0) and (π_b,θ_b) over (C,D) and (C_b,D_b). The two proofs are given back to $\mathcal{A}$ that outputs a bit b' and sends it to $\mathcal{B}$. This latter answers its own challenger $\mathcal{C}$ with the same bit b' . As such, $\mathcal{A}$ succeeds in breaking the unlinkability property of SEVIL with the same probability of breaking the computational witness-indistinguishability property, which is negligible. Thus, SEVIL ensures the unlinkability property w.r.t. the computational witness-indistinguishability property of Groth-Sahai NIWI proofs.

7 PERFORMANCE ANALYSIS

This section first introduces the test environment. Then, it presents the performances' analysis of SEVIL according to the computation time, the complexity and the communication cost of the different algorithms. This evaluation is complemented with (1) a comparative analysis of the computation time of the SEVIL aggregated verification against the naive verification of group signatures, and (2) the evaluation of the impact of the messages' number on the computation time.

7.1 Test Environment

The implementation includes the three phases SETUP, SIGNING and VERIFYING of SEVIL including the six algorithms referred to as Set_params, Setup_SGr$_{\mathcal{G}}$, Join_SGr$_{\mathcal{S}/\mathcal{G}}$, G_Sign$_{\mathcal{S}}$, Batch_Verify$_{\mathcal{V}}$ and Agg_Verify$_{\mathcal{V}}$. For comparison purposes, the four primitives of the group signature scheme presented in Section 4.2 are also implemented relying on the public parameters obtained through the Set_params algorithm.

Our tests are conducted on an Ubuntu 18.04.3 machine - with an *Intel* Core $i7@1.30GHz$ processor and $8GB$ memory. Based on JAVA version 11, the associated cryptographic library *JPBC*[4] and the implementation of Groth-Sahai proofs[5], the SEVIL test-bed is built upon four main java classes, w.r.t. to the different entities of SEVIL, referred to as *TrustedAuthority.java*, *GroupManager.java*, *Signer.java* and *Verifier.java*. For each class, we defined different methods w.r.t. the algorithms performed by each entity as described in Section 3.

For efficiency purpose, two types of improvements are introduced in SEVIL algorithms. The improvements are applied, in particular, on G_Sign and Batch_Verify algorithms of SEVIL and the GSIG.Verify algorithm of GSIG signature scheme, as follows:

- **Multithreading:** applied on the four algorithms G_Sign, Batch_Verify, Agg_Verify and GSIG.Verify. It enables to simultaneous execute multiple threads on different processor cores. The multithreading helps G_Sign to compute different parts of the NIWI proof simultaneously, while for Batch_Verify, Agg_Verify and GSIG.Verify algorithms, it enables higher computation throughput on both sides of the verification equations of the NIWI proof.

[4]http://gas.dia.unisa.it/projects/jpbc/

[5]https://github.com/gijsvl/groth-sahai

- **Preprocessing:** applied only on the Batch_Verify, Agg_Verify and GSIG.Verify algorithms. It helps reduce the computation time when some variables need to be computed several times during the execution of the algorithm. This is the case for the variables (e.g., $\mathtt{U}$ and $\mathtt{V}$ of the CRS Σ_{NIWI}) which can be prepared in advance (i.e., before the execution of algorithms) for next be provided as input to the pairing functions of the Batch_Verify, Agg_Verify and GSIG.Verify algorithms.

Those two improvements are efficient as the computation time can be decreased by up to 50%.

Based on the *JPBC* library, we choose to evaluate the computation time of each algorithm relying on two different types of bilinear pairings, referred to as pairings *type A* and *type F*. Pairing *type A* represents the fastest symmetric pairing type while relying on the elliptic curve $y^2 = x^3 + x$ with an embedding degree equal to 2. Pairing *type F* supports asymmetric pairing features and was introduced by Barreto and Naehrig (pai,) with an embedding degree equal to 12. Two different levels of security are considered for the pairings *type A* and *type F* referred to as 112-bits and 128-bits security levels[6].

The tests rely on 100 samples of randomly generated messages. Each algorithm is run 100 times, and the given computation times are the average of the 100 runs. The standard deviation of an order 10^{-2} is considered.

7.2 Computation Overhead of SEVIL

This section presents the theoretical and experimental computing costs of SEVIL's six algorithms.

Table 1 shows that the G_Sign and Batch_Verify are the most consuming algorithms in terms of exponentiation and pairing operations. To sign a single message, G_Sign requires 302 exponentiations and 18 multiplications in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$. The theoretical computation cost of Batch_Verify mainly depends on the number of messages, especially in terms of multiplication and pairing operations. The significant computation costs of G_Sign and Batch_Verify algorithms are reduced thanks to the two steps of improvements presented in Section 7.1.

Table 1 shows that the selected pairing types along with the security level strongly impact the computation times. Note that both Set_params and Setup_SGr algorithms, as part of the SETUP phase, are consuming but they are limited to only one execution, respectively from a powerful trusted authority and a group manager. The performances of the Join_SGr algorithm depend on the selected pairing type and security level, with respectively 2 and 6 seconds for symmetric pairing settings (i.e., pairing *type A*) for respectively 112 and 128 bit security, and 1,2 and 1,4 seconds for asymmetric pairing settings (i.e., pairing *type F*), for respectively 112 and 128 bit security. For the SIGNING phase, the G_Sign algorithm is also consuming, with respectively 19 and 40 seconds in symmetric pairing settings, and 3 and 4 seconds in asymmetric pairing settings. Finally, for the VERIFYING phase, the Batch_Verify algorithm executed to verify 100 messages simultaneously, requires approximately 4 and 8 minutes for pairing *type A* and 17 and 22 minutes for pairing *type F*. However, when it is needed to verify a single message, the Agg_Verify algorithm requires 3 and 7 seconds for pairing *type A* and 16 and 19 seconds for pairing *type F*. It is worth noticing that, for a number of messages $N = 100$, the execution of the Batch_Verify algorithm gives improved computational costs compared to the Agg_Verify algorithm executed 100 times, separately.

Experimental results depicted in Table 1, confirm the theoretical results that G_Sign, Batch_Verify and Agg_Verify are the most consuming algorithms. This is logical as they include a large number of exponentiations and pairing functions. However this result must be put into perspective as both the signer and the verifier are assumed to be powerful and have advanced hardware features.

From Table 1, it is also clear that the G_Sign algorithm performed with asymmetric pairing settings is faster than with symmetric settings. Indeed, the elementary functions of multiplication and exponentiation are more consuming for pairing *type A* than for pairing *type F*[7]. However the Batch_Verify and Agg_Verify algorithms have an opposite behavior with a faster execution with pairing *type A* than with pairing *type F*. This can be explained by the excessive memory allocation and deallocation needed by pairing *type F*.

7.3 Communication Overhead of SEVIL

This section discusses the communication costs of SEVIL. As shown in Table 1, the communication cost is evaluated according to the size of group elements $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$ and $\mathbb{Z}_n$. Each pairing type and each security level are characterized with different group sizes. From Table 1, it is worth noticing that the SETUP phase is the most consuming in terms

[6]The 112-bits and 128-bits security levels are recommended by the US National Institute of Standards and Technology (NIST) (http://keylength.com).

[7]The experimental results are thus compliant to the *JPBC* library http://gas.dia.unisa.it/projects/jpbc/benchmark.html

Table 1: Computation and communication cost of SEVIL.

Algorithm	Entity	Synch/Asynch	Communication cost	Complexity	Computation time (ms)			
					A/112-bits	A/128-bits	F/112-bits	F/128-bits
Set_params	$\mathcal{TA}$	Asynch.	$\lvert\mathbb{Z}_n\rvert+\lvert\mathbb{G}_1\rvert+\lvert\mathbb{G}_2\rvert+\lvert\mathbb{G}_3\rvert$	γ_G	874	2521	1230	1364
Setup_SGr	$\mathcal{G}$	Asynch.	$21\lvert(\mathbb{G}_1\rvert+\lvert\mathbb{G}_2\rvert)$	$24\gamma_{E_{1,2}}$	1955	4075	346	451
Join_SGr	$\mathcal{S}/\mathcal{G}$	Synch.	$(\mathcal{S})$: $8\lvert\mathbb{G}_1\rvert+2\lvert\mathbb{G}_2\rvert$ / $(\mathcal{G})$: $7(\lvert\mathbb{G}_1\rvert+\lvert\mathbb{G}_2\rvert)$	$(\mathcal{S})$: $6\gamma_{E_{1,2}}$ / $(\mathcal{G})$: $32\gamma_{E_{1,2}}+22\gamma_{M_{1,2}}$	2861	6014	1159	1409
G_Sign [a]	$\mathcal{S}$	Synch.	$6(\lvert\mathbb{G}_1\rvert+\lvert\mathbb{G}_2\rvert)$	$302\gamma_{E_{1,2}}+18\gamma_{M_{1,2}}$	19353	40371	3164	4170
Batch_Verify [b]	$\mathcal{V}$	Asynch.	N.A.	$4\gamma_{E_3}+(6N+10)\gamma_{M_3}+(6N+9)\gamma_P$	222989	485233	1018375	1312879
Agg_Verify	$\mathcal{V}$	Asynch.	N.A.	$4\gamma_{E_3}+16\gamma_{M_3}+15\gamma_P$	3096	6916	16065	18834

NOTE: Synch./Asynch. indicates whether the algorithm must be run online (i.e. in real time) or offline (i.e. later); [a] indicates that the algorithm is performed on a single message; [b] indicates that the algorithm is performed on N messages where $N=100$ for computation times; $|\mathbb{G}_1|$ (resp. $|\mathbb{G}_2|$, $|\mathbb{G}_3|$ and $|\mathbb{Z}_n|$) indicates the size of an element in $\mathbb{G}_1$ (resp. $\mathbb{G}_2$, $\mathbb{G}_3$ and $\mathbb{Z}_n$); γ_G is the cost of the cyclic group generation; $\gamma_{M_{1,2}}$ and γ_{M_3} are the costs of multiplication in resp. $\mathbb{G}_1/\mathbb{G}_2$ and $\mathbb{G}_3$; $\gamma_{E_{1,2}}$ and γ_{E_3} are the costs of exponentiation in resp. $\mathbb{G}_1/\mathbb{G}_2$ and $\mathbb{G}_3$; γ_P is the cost of a pairing function; N.A. is the abbreviation for Not Applicable.

of bandwidth. In fact, it includes the Set_params and Setup_SGr algorithms that output the system and group public parameters shared with other entities. The SETUP phase also includes the interactive Join_SGr algorithm that introduces communication overheads of $8|\mathbb{G}_1|+2|\mathbb{G}_2|$ and $7(|\mathbb{G}_1|+|\mathbb{G}_2|)$ to respectively send the signer's keys to the $\mathcal{G}$ and give back $\mathcal{G}$'s signature over the keys of $\mathcal{S}$. The communication cost introduced by the SETUP phase must be put into perspective as both algorithms Set_params and Setup_SGr are executed once, and the Join_SGr algorithm is performed only when a new signer wants to join the group. The SIGNING phase, including only the G_Sign algorithm repeatedly performed by signers, has an acceptable communication overhead due to the size of the NIWI proof.

7.4 Benefit of SEVIL Aggregated Verification over GSIG Naive Verification

In this section, we focus on the VERIFYING phase. We consider 100 messages signed with the G_Sign algorithm. The resulting proofs are given as input to both GSIG.Verify and Batch_Verify algorithms. The GSIG.Verify algorithm is executed 100 times as it allows to verify only one proof at a time, while the Batch_Verify algorithm performs the verification of all proofs at a time. Thus, we compare the computation time required by the two algorithms when being executed over 100 messages.

Figure 5 shows that the aggregated verification is more efficient than the naive one. Indeed, the aggregation reduces the computation time, for verifying 100 messages, by approximately 37%, for pairing *type A* for the two security levels. The computation time moves from 356 seconds (resp. 777 seconds) with the naive signature verification to 223 seconds (resp. 485 seconds) with SEVIL aggregated verification. For pairing *type F*, the gain reaches 50% for

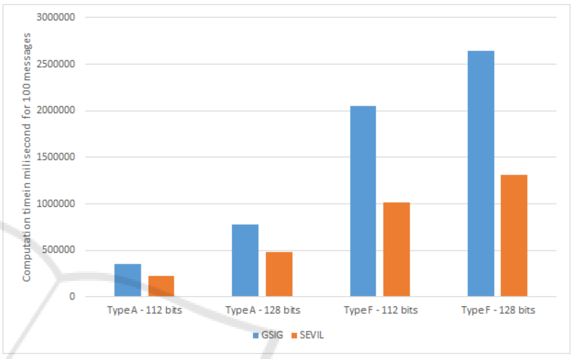

Figure 5: Computation time of aggregated verification vs naive verification over 100 messages.

the two security levels. The computation time moves from 2048 seconds (resp. 2642 seconds) to 1018 seconds (resp. 1313 seconds).

The gain obtained through the aggregating verification is substantiated by the decrease in the number of pairings. To verify N messages (i.e., 6 NIWI proofs are verified per messages), the GSIG.Verify algorithm requires $30N$ pairings (according to Equation 1), while the Batch_Verify algorithm only requires $6N+9$ pairings. Thus, we expect to obtain a gain of approximately 80%, but experimental results show smaller gains than expected. These results are justified by the number of additions introduced while aggregating the verification equations (i.e., $14N$ additions). As mentioned before, the elementary addition operations are more consuming for pairing *type A* than for pairing *type F*. Hence, the gain is more significant with asymmetric pairing settings.

7.5 Impact of Messages' Volume on the Verification

Referring to equations (2) and (3), depicted in Figure 3, it is clear that the number N of messages (resp. proofs) to be verified, influences the time computation of the Batch_Verify algorithm. Indeed, the greater the number of messages, the greater the number of pairing functions. For this objective, we evaluate the computation time of the Batch_Verify algorithm when varying the number of messages from 5 to 1000. Note that all messages are signed with the G_Sign algorithm.

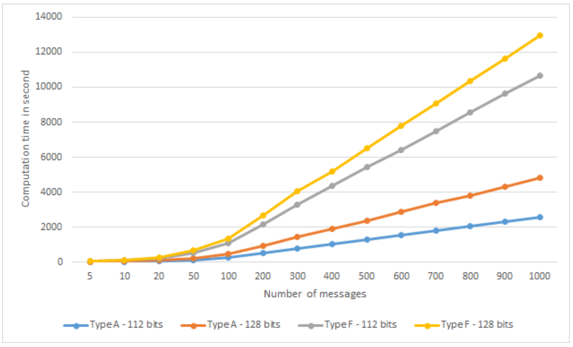

Figure 6: Influence of messages' volume on Batch_Verify computation time.

The curves depicted in Figure 6 show that the computation time of the Batch_Verify algorithm is a rising affine function of the messages number, for the two types of pairings and the two security levels. When varying the number of messages from 5 to 1000, the computation time varies from 15 to 2602 seconds (resp. from 59 to 10677) for the pairing *type A* (resp. pairing *type F*), for 112-bits level. For the 128-bits security, the computation time varies from 26 to 4817 (resp. 72 to 12978) for the pairing *type A* (resp. pairing *type F*).

8 COMPARISON WITH RELATED WORK

Data-centric applications, e.g., cloud-based technologies (di Vimercati et al., 2019), recommendation systems (Kaaniche et al., 2020b; Rahali et al., 2021) and IoT applications (Alamer, 2020; Zhang et al., 2021; Belguith et al., 2018), have raised several concerns regarding the massive collection, processing and access to data from different users with different privileges (Kaaniche et al., 2020a). Several works have been proposed, in the literature, to efficiently verify a large number of signatures, referred to as signature schemes with batch verification. This method helps to solve the resource constraints' problems in many applications. Batch verification for signatures was first proposed by Naccache *et. al* (Naccache et al., 1994) for DSA-type signatures. Since then, several batch verification methods have been proposed for other digital signature schemes, namely for group signatures. Indeed, batch verification over group signatures was introduced by Ferrara *et. al* (Ferrara et al., 2009). Wasef and Shen proposed to use batch verification. Vehicular ad hoc networks (Wasef and Shen, 2010). In (Kim et al., 2011), authors exploited Ferrara *et. al* scheme to build a new batch vertification scheme that supports invalid signatures identification. They rely on the divide-and-conquer approach (Pastuszak et al., 2004). In (Feng et al., 2017), authors presented a group signature scheme with batch verification that allows to deal with the excessive need for signatures verification in pervasive social networking. The proposed scheme do not support bad signatures identification, i.e., if the batch verification fails, all the signatures are rejected. Recently, Alamer proposed a secure and privacy-preserving group signature scheme supporting batch verification. It aims at mitigating the increasing computation delay in IoT systems (Alamer, 2020). In (Zhang et al., 2021), authors designed a novel group signature scheme with batch verification for IoT consortium blockchain. It suggests two types of verification, i.e., a naive verification for urgent transactions and batch verification for ordinary ones.

Table 2 illustrates differences between SEVIL and closely related schemes in terms of security and privacy properties and supported functionalities.

From Table 2, it is worth stating that SEVIL satisfies several properties of interest, compared to closely related proposals. It leverages the trade-off between security, privacy and utility. Indeed, as all others schemes, SEVIL fulfills the unforgeability requirement. Unlike (Wasef and Shen, 2010), (Kim et al., 2011), (Alamer, 2020) and (Zhang et al., 2021), the proposed scheme adds security features, referred to as trust on signers. SEVIL also addresses a critical privacy concern which is unlinkability. In terms of utility, we note that, unlike (Wasef and Shen, 2010), (Feng et al., 2017), (Alamer, 2020) and (Zhang et al., 2021) which only support batch verification, the SEVIL system is designed to support identification of invalid signatures, which is very relevant for information accuracy.

9 CONCLUSION

In this paper, we introduce a concrete construction of a novel secure and privacy-preserving Groth-Sahai NIWI proof-based signature. The proposed scheme enables the efficient verification of multiple signatures, i.e. allowing verifiers to check the correctness of multiple proof-based group signatures, at once. Our contribution is proven to support security and

Table 2: Comparison of SEVIL and related works.

		SEVIL	(Wasef and Shen, 2010)	(Kim et al., 2011)	(Feng et al., 2017)	(Alamer, 2020)	(Zhang et al., 2021)
Security and privacy properties	Unforgeability	✓	✓	✓	✓	✓	✓
	Trust on signers	✓	✗	✗	✓	✗	✗
	Unlinkability	✓	✓	✗	✗	✓	✓
Functional properties	Batch verification	✓	✓	✓	✓	✓	✓
	Invalid signatures identification	✓	✗	✓	✗	✗	✗

privacy properties, through a comprehensive security analysis. Thanks to SEVIL's proof of concept that fully implement the different algorithms, we show that the aggregated verification achieves a gain of up to 50% with regard to the naive verification of group signatures. This gain proves the efficiency of SEVIL and must be put into perspective as in real world applications, verifiers are assumed to have advanced hardware features.

ACKNOWLEDGEMENTS

Authors are thankful to Mr. Fadel Radji for implementing SEVIL's algorithms and for providing valuable suggestions to improve the system's performances.

REFERENCES

Jpbc library: Bilinear pairing parameters generators. http://gas.dia.unisa.it/projects/jpbc/docs/ecpg.html.

Alamer, A. (2020). An efficient group signcryption scheme supporting batch verification for securing transmitted data in the internet of things. *Journal of Ambient Intelligence and Humanized Computing*.

Belguith, S., Kaaniche, N., Mohamed, M., and Russello, G. (2018). Coop-daab: Cooperative attribute based data aggregation for internet of things applications. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 498–515. Springer.

Bellare, M., Micciancio, D., and Warinschi, B. (2003). Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Biham, E., editor, *Advances in Cryptology — EUROCRYPT 2003*, pages 614–629, Berlin, Heidelberg. Springer Berlin Heidelberg.

di Vimercati, S. D. C., Foresti, S., Livraga, G., and Samarati, P. (2019). Data security and privacy in the cloud. In Agaian, S. S., Asari, V. K., and DelMarco, S. P., editors, *Mobile Multimedia/Image Processing, Security, and Applications 2019*, pages 84 – 96. SPIE.

Feng, W., Yan, Z., and Xie, H. (2017). Anonymous authentication on trust in pervasive social networking based on group signature. *IEEE Access*, 5:6236–6246.

Ferrara, A., Green, M., Hohenberger, S., and Pedersen, M. (2009). Practical short signature batch verification. pages 309–324.

Groth, J. and Sahai, A. (2008). Efficient non-interactive proof systems for bilinear groups. In Smart, N., editor, *Advances in Cryptology – EUROCRYPT 2008*, pages 415–432, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kaaniche, N., Laurent, M., and Belguith, S. (2020a). Privacy enhancing technologies for solving the privacy-personalization paradox: Taxonomy and survey. *Journal of Network and Computer Applications*, page 102807.

Kaaniche, N., Masmoudi, S., Znina, S., Laurent, M., and Demir, L. (2020b). Privacy preserving cooperative computation for personalized web search applications. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, page 250–258.

Kim, K., Yie, I., Lim, S., and Nyang, D. (2011). Batch verification and finding invalid signatures in a group signature scheme. *International Journal of Network Security*, 12:229–238.

Naccache, D., M'Raïhi, D., Vaudenay, S., and Raphaeli, D. (1994). Can d.s.a. be improved? complexity trade-offs with the digital signature standard. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994*, Lecture Notes in Computer Science, pages 77–85. Springer.

Pastuszak, J., Michałek, D., Pieprzyk, J., and Seberry, J. (2004). Identification of bad signatures in batches. pages 28–45.

Rahali, S., Laurent, M., Masmoudi, S., Roux, C., and Mazeau, B. (2021). A validated privacy-utility preserving recommendation system with local differential privacy.

Wasef, A. and Shen, X. (2010). Efficient group signature scheme supporting batch verification for securing vehicular networks. In *2010 IEEE International Conference on Communications*, pages 1–5.

Zhang, A., Zhang, P., Wang, H., and Lin, X. (2021). Application-oriented block generation for consortium blockchain-based iot systems with dynamic device management. *IEEE Internet of Things Journal*, 8(10):7874–7888.