Backward Unlinkability for a VLR Group Signature Scheme with Efficient Revocation Check

Julien Bringer[1] and Alain Patey[1,2]

[1] Morpho
[2] Télécom ParisTech

Abstract. Verifier-Local Revocation (VLR) group signatures, introduced by Boneh and Shacham in 2004 are a particular case of dynamic group signature schemes where the revocation process does not influence the activity of the signers. The verifiers use a Revocation List to check if the signers are revoked. In all known schemes, checking a signature requires a computational time linear in the number of revoked members. Usually, it requires one pairing per revoked user. Recently, Chen and Li proposed a scheme where Revocation Check uses exponentiations instead of pairings. Moreover, their scheme is aimed to offer protection against the Group Manager by enabling Exculpability. In this paper, we propose a correction of their scheme to enable a full proof of the traceability property and an extension to ensure Backward Unlinkability. This property prevents the loss of anonymity of past signatures when a user is revoked. This improvement requires a constant additional cost to the signature size whereas revocation check remains as efficient. We thus obtain the scheme with the most efficient Revocation Check among VLR schemes enabling Backward Unlinkability.

Keywords: group signatures, verifier-local revocation, backward unlinkability, exculpability, efficiency, revocation check

1 Introduction

Group signatures, introduced by Chaum and van Heyst [12], enable a registered member to sign anonymously on behalf of a group. The identity of a signer can only be revealed by a Group Manager who knows all the secret parameters of the group. Actual group signature schemes are dynamic: members can join and leave (voluntarily or not) the group at any time. To enable this, a revocation process is established. First dynamic schemes proposed to reissue new keys for everyone but the revoked members. In the same spirit, some schemes offer to use dynamic accumulators [10].

We focus in this paper on schemes with *Verifier-Local Revocation* (VLR), a particular way to deal with revocation where we do not want additional interactions with the signers. A Revocation List (RL) is built by the group manager

and sent only to the verifiers. The signers do not take it into account when they sign. Verifying a signature is divided into two parts: a *Signature Check* to verify if the signer is a registered member and a *Revocation Check* to verify, using RL, whether the signer is revoked. This type of group signature schemes is useful for applications where signers are often offline or are computationally weak devices (TPMs, smartcards...). Several proposals for VLR group signatures have been made, either in the Random Oracle Model as [6,13,17–19,26–29] or in the Standard Model as [14,16]. A similar concept is introduced in [15] for traceable signatures with an implicit tracing mechanism. Applications of VLR schemes are, for instance, Direct Anonymous Attestation (DAA) in the context of Trusted Computing [7, 8], Vehicular Ad-hoc NETworks (VANETs) [25] or anonymous authentication [9].

One downside of VLR schemes is the lack of efficiency of the Revocation Check during the verification of a signature. Indeed, in the original [6] scheme and many other propositions like [16,18], this part requires at least one pairing operation per each revoked user. [19] proposed a slight variant where products of pairings are used instead of separate pairings. In [13], Chen and Li proposed a VLR scheme using exponentiations in the Revocation Check. As exponentiations require less computation time, this is a substantial improvement concerning efficiency. Another VLR-like scheme based on exponentiations is described in [28] but it has weaker security properties (no coalition-resistance).

However, in the [13] scheme, it is unclear how to obtain an extractor for the proof of knowledge included in the signature. Having an extractor is necessary for the proof of *traceability*, one of the essential security properties required from a group signature scheme. This is why we propose a patch to the original [13] scheme and explain explicitly how to build an extractor for the thus modified algorithm.

Another issue in most VLR schemes is the following: once a user has been revoked, all his previous signatures loose their anonymity. The property that prevents this loss is called *Backward Unlinkability* (BU). It also allows a user to come back into the group after having been revoked and use the same keys as before while remaining anonymous. This property was first introduced by Song in [22]. There have been several proposals to enable BU in schemes using pairings in the Revocation Check, *e.g.* [18] in the Random Oracle Model or [16] in the Standard Model. This does not change the type of operations to use in the Revocation Check. The other parts of the signing and verifying algorithms are slightly modified but the difference is constant and small.

The same techniques cannot be applied to schemes based on exponentiations. A first proposal for such schemes has been suggested without specific proofs in [2] in the context of quadratic residues, based on the [1] scheme. We present in this paper an improvement, inspired by the technique from [2] that we adapt to the context of bilinear groups, to the efficient [13] scheme to add the BU property. Moreover we obtain full proofs of our security results included the BU functionality and we also patch the [13] scheme in order to ensure traceability. To achieve BU, we use zero-knowledge proofs of knowledge involving double discrete

logarithms. This technique requires a number of computations that is a function of a security parameter, but that is independent from the total number of users and from the number of revoked members. Moreover, this technique is generic and can be applied to other exponentiation-based VLR schemes, *e.g.* [28].

Our scheme satisfies *Backward Unlinkability*, *Traceability* and *Exculpability* in the random oracle model. Security is based on the strong Diffie-Hellman (SDH) assumption, a slight modification of the Decisional Diffie-Hellman (modified DDH) assumption and the Discrete Logarithm (DL).

In the sequel, we first give formal definitions for VLR schemes and their security requirements in Section 2. We then describe the mathematical background needed to build our scheme in Section 3. We describe in Section 4 our proposal to add BU to exponentiation-based VLR schemes based on the [13] suggestion. We prove its security in Section 6 and analyze its efficiency in Section 7. We finally conclude in Section 8.

2 Definitions

In this section, we describe the model for VLR group signature schemes and the security properties that we expect from our scheme. We extend the VLR group signature model from [13] by including Backward Unlinkability following the model of [18]. The security properties for generic dynamic group signatures are from [4, 6] for Verifier-Local Revocation and from [18] to enable Backward Unlinkability.

2.1 VLR Group Signatures

There are three types of entities in our model: a Group Manager GM, a set of members and a set of verifiers.

A VLR Group Signature Scheme with Backward Unlinkability and Exculpability consists of the following algorithms:

KeyGen(k, T)**:** On input a security parameter k and a number T of periods, this algorithm, run by GM outputs the group public key gpk and the issuing key ik. It also sets an empty Revocation List RL_j, for each period j. These lists will be filled later with the revocation tokens of the revoked users.

Join$(gpk, ik; gpk)$**:** This algorithm is an interactive protocol between GM and a member M_i. GM takes as input the public parameters gpk and the issuing key ik, M_i takes only gpk.
In the end, M_i outputs an identity id_i, a secret key sk_i, a credential cre_i and a tracing key tk_i (included in cre_i). GM gets id_i and tk_i and outputs also a list of revocation tokens for M_i: $\mathbf{rt_i} = \{rt_{ij} | j \in \{1, \ldots, T\}\}$.

Revoke(gpk, rt_{ij}, j, RL_j)**:** GM runs this algorithm to prevent a member M_i from making valid signatures at period j. It outputs an updated revocation list RL_j for period j, where rt_{ij} has been added.

Sign(gpk, j, sk_i, cre_i, m)**:** This algorithm, run by a member M_i, takes as input a message m, M_i's keys sk_i and cre_i and a message m to sign at period j. It outputs a signature σ.

Verify$(gpk, j, RL_j, m, \sigma)$**:** This algorithm, run by a verifier takes as input a message m, its signature σ, a period j, the corresponding Revocation List RL_j and the public parameters gpk. It checks if the message has been signed by an unrevoked group member, without revealing the signer's identity. The possible outputs are **valid** and **invalid**.

Open$(gpk, j, m, \sigma, \{tk_i\}_i)$**:** This algorithm is run by GM. It takes a signature σ on a message m at period j as input, together with all tracing keys of the group. It reveals the tracing key tk_i and the identity id_i of the signer. If it fails to recover the identity of the signer, it returns $\perp$.

2.2 Security Requirements

We require our scheme to satisfy *Correctness*, *Backward Unlinkability*, *Traceability* and *Exculpability*. These properties are described below.

Correctness The scheme is *correct* if every signature created by an unrevoked member is verified as valid. Formally, $\forall \sigma = \mathit{Sign}(gpk, j, sk_i, cre_i, m)$, $(\mathit{Verify}(gpk, j, RL_j, m, \sigma) = \textbf{valid}) \Leftrightarrow rt_{ij} \notin RL_j$.

Backward Unlinkability This model of *Backward Unlinkability* applies BU to the case of Selfless-Anonymity[3]. Consider the following BU game played by an adversary $\mathcal{A}$:

Setup: The challenger $\mathcal{C}$ runs $\mathit{KeyGen}(k)$ and obtains gpk and ik. He sends gpk to $\mathcal{A}$.

Queries: At the beginning of each period j, $\mathcal{C}$ announces the beginning of j to $\mathcal{A}$, they both increment j simultaneously. At the current time period j, $\mathcal{A}$ can run the following queries:

- **Join:** $\mathcal{A}$ requests the creation of a new member. $\mathcal{C}$ runs $Join$, obtains a new pair (sk_i, cre_i), stores it but does not send it to $\mathcal{A}$.
- **Sign:** $\mathcal{A}$ requests a signature on a message m by a member M_i. $\mathcal{C}$ computes $\sigma = \mathit{Sign}(gpk, j, sk_i, cre_i, m)$ and sends it to $\mathcal{A}$.
- **Corruption:** $\mathcal{A}$ requests the signing key of a member M_i. $\mathcal{C}$ sends (sk_i, cre_i) to $\mathcal{A}$.
- **Revocation:** $\mathcal{A}$ asks for revocation of a member M_i for period j. $\mathcal{C}$ returns rt_{ij}, adds it to RL_j, sends it to $\mathcal{A}$.

Challenge: $\mathcal{A}$ outputs a message m^* and two members M_{i_0} and M_{i_1}, who are neither corrupted, nor revoked at current time j^*. $\mathcal{C}$ chooses $b \in_R \{0, 1\}$ and sends $\sigma^* = \mathit{Sign}(gpk, j^*, sk_{i_b}, cre_{i_b}, m^*)$ to $\mathcal{A}$.

Restricted Queries: $\mathcal{A}$ can make the same queries as in the *Queries* phase, as long as he requires neither the revocation for period j^* or the corruption of M_{i_0} or M_{i_1} nor the opening of σ^*.

[3] Unlike *Full-Anonymity* which prevents signatures from giving any information about their signer, in the *Selfless-Anonymity* case as defined by [6], a member knows if a given signature was signed by him or not, but he does not gain any other information.

Output: $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$ on b.

The scheme is said to satisfy the *Backward Unlinkability* property if the probability $|\Pr[b = b'] - 1/2|$ is negligible for any probabilistic polynomial-time adversary $\mathcal{A}$.

Traceability The scheme is *traceable* if an adversary $\mathcal{A}$ can forge a valid signature that cannot be opened properly. Consider the following *Traceability* game played by $\mathcal{A}$:

Setup: The challenger $\mathcal{C}$ runs $KeyGen(k)$ and obtains gpk and ik. He sends gpk to $\mathcal{A}$. He also sets an empty revocation list RL.

Queries: At the beginning of each period j, $\mathcal{C}$ announces the beginning of j to $\mathcal{A}$, they both increment j simultaneously. At the current time period j, $\mathcal{A}$ can execute the following queries:

- **Join:** $\mathcal{A}$ requests the creation of a new member M_i at current period j. There are two possibilities:
  1. $\mathcal{C}$ runs *Join* alone, obtains a new pair (sk_i, cre_i), stores it but does not send it to $\mathcal{A}$.
  2. $\mathcal{C}$ and $\mathcal{A}$ interact, $\mathcal{C}$ playing the role of the issuer and $\mathcal{A}$ the role of the new member. $\mathcal{A}$ obtains his secret key and credential. $\mathcal{C}$ gets only the identity id_i and the tracing key tk_i. This user is directly revoked: the revocation tokens $rt_{ij'}$ of M_i are added to $RL_{j'}$ for every $j' \geq j$.
- **Corrupt:** $\mathcal{A}$ requests sk_i and cre_i of a member M_i, at the current period j. $\mathcal{C}$ gives them to him and revokes M_i: the revocation tokens $rt_{ij'}$ of M_i are added to $RL_{j'}$ for every $j' \geq j$.
- **Sign:** Same as in the *Backward Unlinkability* game.

Output: $\mathcal{A}$ outputs a message m^*, the current interval j^*, the Revocation List RL_{j^*} and a signature σ^*.

$\mathcal{A}$ wins the game if:

1. $Verify(gpk, j^*, RL_{j^*}, m^*, \sigma^*) = valid$ (implying that σ^*'s opening traces to a member outside the coalition);
2. and $\mathcal{A}$ did not obtain σ^* by making a Sign query on m^*.

The scheme satisfies *Traceability* if no polynomial probabilistic adversary is able to win the above game with a non-negligible probability.

Exculpability In the *Exculpability* game, roles are inverted: the adversary is the GM and, consequently, knows the group's secret key and all the players' credentials. The goal of the adversary is to forge a valid signature that will be attributed to an honest (*i.e.* not corrupted) member by the *Open* algorithm. This signature must be such that it cannot be denied by the signer. Consider the following *Exculpability* game played by an adversary $\mathcal{A}$:

Setup: $\mathcal{C}$ runs $KeyGen(k, T)$ and obtains gpk and ik. $\mathcal{C}$ stores gpk and sends gpk and ik to $\mathcal{A}$[4].

Queries: At the beginning of each period j, $\mathcal{C}$ announces the beginning of j to $\mathcal{A}$, they both increment j simultaneously. At the current time period j, $\mathcal{A}$ can request the following queries:

[4] This corresponds to a trusted setup run on behalf of the adversary.

- **Join:** $\mathcal{A}$ requests the creation of a new member M_i at current period j. $\mathcal{A}$ plays the role of the issuer and $\mathcal{C}$ the role of the member. $\mathcal{C}$ gets a signing key (id_i, sk_i, cre_i) and $\mathcal{A}$ gets (id_i, tk_i)
- **Corrupt:** $\mathcal{A}$ requests sk_i and cre_i of a member M_i, at the current period j. $\mathcal{C}$ gives them to him and revokes M_i: the revocation tokens $rt_{ij'}$ of M_i are added to $RL_{j'}$ for every $j' \geq j$.
- **Sign:** Same as in the *Backward Unlinkability* game.

Output: $\mathcal{A}$ outputs a message m^*, the current interval j^*, the corresponding Revocation List RL_{j^*},a signature σ^* and a tracing key pair (id_{i^*}, tk_{i^*})

$\mathcal{A}$ wins the game if:

1. he did not obtain σ^* from making a query on m^*;
2. $Verify(gpk, j^*, RL_{j^*}, m^*, \sigma^*) = \textbf{valid}$;
3. $Open(gpk, j^*, m^*, \sigma^*, \{tk_i\}) = (id_{i^*}, tk_{i^*})$;
4. he did not corrupt M_{i^*};
5. and it is impossible for M_{i^*} to prove the knowledge of a member key (id_i, sk_i, cre_i) such that $sk_i \neq sk_{i^*}$ and such that sk_i could have issued σ^* as a valid signature[5].

Note that the last requirement is to ensure protection against a dishonest opening from the manager. The scheme satisfies *Exculpability* if no polynomial probabilistic adversary is able to win the above game with a non-negligible probability.

3 Preliminaries

In this section, we introduce some notations and the complexity assumptions on which our scheme relies. We then explain how to manage a proof of knowledge of an equality between a discrete logarithm and a double discrete logarithm. We use this technique as a basis to extend [13] scheme to Backward Unlinkability.

3.1 Bilinear Groups and Pairings

Let G_1 be a cyclic group of prime order p, G_2 be a group of order a power of p, G_T be a cyclic group of prime order p, ψ be an homomorphism from G_2 to G_1, g_2 be an order-p element of G_2 and g_1 a generator of G_1 such that $\psi(g_2) = g_1$.

A *pairing* is a map $e : G_1 \times G_2 \to G_T$ that is *bilinear* ($\forall u \in G_1, v \in G_2, a, b \in \mathbb{Z}, e(u^a, v^b) = e(u, v)^{ab}$) and *non-degenerate* ($e(g_1, g_2) \neq 1$). We will refer to such groups G_1, G_2 and G_T as *bilinear groups*.

3.2 Complexity Assumptions

The security of our scheme relies on the DL assumption, the q-SDH assumption [5] and a modification of the DDH assumption. We define them below.

[5] This condition is described in [13] by using two functionalities called DProve and DVerify.

Discrete Logarithm (DL) problem
Given: G a multiplicative finite cyclic group, with generator g, and g^n (with $n \in_R \mathbb{Z}$).
Problem: Find n.

q-Strong Diffie-Hellman (q-SDH) problem
Given: bilinear groups G_1, G_2, G_T, g_1, g_2 and a pairing e, as in Section 3.1, and a q-tuple $(g_2^{\gamma}, .., g_2^{(\gamma^q)})$ $(\gamma \in_R \mathbb{Z}_p^*)$.
Problem: Compute a pair $(g_1^{1/(\gamma+x)}, x)$, with $x \in \mathbb{Z}_p^*$

Modified DDH problem Given: G a multiplicative finite cyclic group of order p, with generator g, g^a, g^b $(a, b \in_R \mathbb{Z}_p^*)$, u a generator of $\mathbb{Z}_p$ and u^a.
Problem: Distinguish g^{ab} from a random element $z \in G$.

3.3 Proofs of Knowledge for Double Discrete Logarithms

Let G be a cylic group of prime order p, $g \in G$, $h \in \mathbb{Z}_p$ and $x \in \mathbb{Z}_p^*$. Let $K = g^x$ and $L = g^{h^x}$. We want to build a Non-Interactive Zero-Knowledge Proof of Knowledge of x.

Unfortunately, a NIZK PK *à la* Schnorr is not possible. Let us recall how it works to prove the knowledge of a discrete logarithm. We consider the same elements but we only prove, given g and K the knowledge of x such that $K = g^x$. We use a hash function $H : \{0,1\}^* \to \mathbb{Z}_p$. For a message m, the prover chooses $r \in_R \mathbb{Z}_p$ and computes $R = g^r$, $c = H(m||K||R)$ and $s = r + cx$. He returns g, K, c, s. To check the proof, the verifier computes $R' = g^s K^{-c}$ then $c' = H(m||K||R')$. If $c = c'$, he accepts the proof, else he rejects it.

It is easy to see that the "trick" used in this proof cannot be reapplied to a double logarithm. We must use binary challenges instead of a modular integer. This technique is due to Camenisch and Stadler [11, 23] and has been suggested for group signatures by Ateniese *et al.* [2].

We describe in Table 1 how such a proof works for a security parameter λ (we keep the same notations for g, h, x, K and L). In [23], the authors state that an attacker can cheat successfully only with probability $2^{-\lambda}$.

Proof Generation:

1. For $l = 1 \ldots \lambda$, pick $r_l \in_R \mathbb{Z}_p$ and compute $V_l = g^{r_l}$, $W_l = g^{h^{r_l}}$.
2. Compute $c = H(m||K||L||(V_l, W_l)_{l=1,\ldots,\lambda})$.
3. For $l = 1 \ldots \lambda$, let b_l denote the lth bit of c. Set $s_l = r_l - b_l x$.
4. Return g, K, L, c, s_1,. . . ,s_λ.

Proof Verification:

1. For $l = 1 \ldots \lambda$, let b_l denote the lth bit of c. Compute $V_l' = g^{s_l} K^{b_l}$ and $W_l' = (g^{1-b_l} L^{b_l})^{h^{s_l}}$
2. Compute $c' = H(m||K||L||(V_l', W_l')_{l=1,\ldots,\lambda})$
3. If $c = c'$, accept the proof, else reject it

Table 1. Proof of knowledge of the equality of a logarithm and a double logarithm.

4 Proposed Scheme

In this section we describe our extension of [13] to prove traceability and to achieve Backward Unlinkability. We discuss also the modification for the traceability proof in Section 5.

The *KeyGen* algorithm is described in Algorithm 1: we use bilinear groups and the notations of Section 3.1 (G_1, G_2, G_T, e, g_1, g_2). The issuing key is $\gamma \in_R \mathbb{Z}_p$, its public counterpart is $w = g_2^{\gamma}$.

Algorithm 1 *KeyGen*(k, T)

1: Choose bilinear groups G_1, G_2, G_T of order a k-bit prime number p that is safe (i.e. $(p-1)/2$ prime number) and a pairing $e : G_1 \times G_2 \to G_T$. Let g_1, g_2 be generators of G_1 and G_2.
2: Choose a hash function $H : \{0,1\}^* \to \mathbb{Z}_p$ and a security parameter λ for the proofs of knowledge involving double logarithms.
3: Choose $\tilde{g}_1, \hat{g}_1 \in_R G_1, \gamma, h_1, \ldots, h_T \in_R \mathbb{Z}_p^*$ of order $(p-1)/2$ and compute $w = g_2^{\gamma}$.
4: Compute $T_1 = e(g_1, g_2)$, $T_2 = e(\tilde{g}_1, g_2)$, $T_3 = e(\hat{g}_1, g_2)$ and $T_4 = e(\hat{g}_1, w)$.
5: Output: $gpk = (G_1, G_2, G_T, e, p, g_1, g_2, \tilde{g}_1, \hat{g}_1, w, H, T_1, T_2, T_3, T_4, \lambda, h_1, \ldots, h_T)$ and $ik = \gamma$.

The *Join* algorithm is explained in Algorithm 2. Each member M_i chooses a secret key $sk_i = f_i \in_R \mathbb{Z}_p$, not known by GM. M_i gives to GM an identity $id_i = \tilde{g}_1^{f_i}$ and proves the knowledge of f_i. GM gives him a credential $cre_i = (A_i, x_i)$, which is almost an SDH couple [5]. It satisfies $e(A_i, wg_2^{x_i}) = e(g_1\tilde{g}_1^{f_i}, g_2)$. To enable Backward Unlinkability, we divide the time into T periods. For each period j, there is a public token h_j. The revocation token for a member M_i at period j is $rt_{ij} = h_j^{x_i}$ and $tk_i = x_i$ is the tracing key. Revocation lists are different at each period, they are denoted RL_j.

Algorithm 2 *Join*$(gpk, ik\ ;\ gpk)$

1: GM sends a nonce $n_i \in \{0,1\}^t$ to M_i.
2: M_i chooses $f_i \in_R \mathbb{Z}_p$ and computes $F_i = \tilde{g}_1^{f_i}$. He sets $sk_i = f_i$ and $id_i = F_i$. He chooses $r_f \in_R \mathbb{Z}_p$ and computes $R = \tilde{g}_1^{r_f}$. He computes $c = H(gpk||F_i||R||n_i)$ then $s_f = r_f + cf_i$.
3: M_i sends $comm = (F_i, c, s_f)$ to GM.
4: GM computes $R' = \tilde{g}_1^{s_f} F_i^{-c}$ and checks that $s_f \in \mathbb{Z}_p$ and $c = H(gpk||F||R'||n_i)$. He chooses $x_i \in_R \mathbb{Z}_p$ and computes $A_i = (g_1 F_i)^{1/(x_i+\gamma)}$. He sets $cre_i = (A_i, x_i)$, $tk_i = x_i$ and $id_i = F_i$.
5: GM sends cre_i to M_i.
6: M_i checks that $e(A_i, wg_2^{x_i}) = e(g_1\tilde{g}_1^{f_i}, g_2)$ and outputs (id_i, sk_i, cre_i).
7: The revocation token for M_i at period j is $rt_{ij} = h_j^{x_i}$.

The *Sign* algorithm is described in Algorithm 3. When a member M_i creates a signature, he first chooses a random $B \in_R G_1$ and computes $J = B^{f_i}$, $K = B^{x_i}$ and $L = B^{h_j^{x_i}}$. He then does a NIZK PK of (f_i, A_i, x_i) satisfying $J = B^{f_i}$, $K = B^{x_i}$ and $e(A_i, w g_2^{x_i}) = e(g_1 \tilde{g}_1^{f_i}, g_2)$. He also provides evidence that $b = ax_i$ as in [6] to ensure traceability based on an extractor (cf. proof of Proposition 2). He finally computes a Proof of Knowledge $(c, (V_l, W_l)_{l=1\ldots\lambda})$, as described in Section 3.3, of the equality: $log_B K = log_{h_j}(log_B L) \quad (= x_i)$.

Algorithm 3 *Sign*(gpk, sk_i, cre_i, m, j)

1: Choose $B \in_R G_1$ and compute $J = B^{f_i}$, $K = B^{x_i}$ and $L = B^{h_j^{x_i}}$.
2: Choose $a \in_R \mathbb{Z}_p$, compute $b = ax_i$ and $T = A_i \hat{g}_1^a$.
3: Choose $r_f, r_x, r_a, r_b, r_1, \ldots, r_\lambda \in_R \mathbb{Z}_p$.
4: Compute $R_1 = B^{r_f}$, $R_2 = B^{r_x}$, $R_4 = K^{r_a} B^{-r_b}$, $V_l = B^{r_l}$, $R_3 = e(T, g_2)^{-r_x} T_2^{r_f} T_3^{r_b} T_4^{r_a}$ and $W_l = B^{h_j^{r_l}}$, $\forall l = 1 \ldots \lambda$.
5: Compute $c = H(gpk||B||J||K||L||T||R_1||R_2||R_3||R_4||j||m)$.
6: Compute $d = H(c||(V_l, W_l)_{l=1\ldots\lambda})$.
7: Compute $s_f = r_f + cf_i$, $s_x = r_x + cx_i$, $s_a = r_a + ca$ and $s_b = r_b + cb$.
8: $\forall l = 1 \ldots \lambda$, let b_l be the lth bit of d. Set $s_l = r_l - b_l.x$.
9: Output: $\sigma = (B, J, K, L, T, c, d, s_f, s_x, s_a, s_b, s_1, \ldots, s_\lambda)$.

The *Verify* algorithm is described in Algorithm 4. To check a signature, the verifier checks both proofs of knowledge. If the checks succeed, he also does a Revocation Check: $\forall rt_{i'j} = h_j^{x_i'} \in RL_j$, he checks that $L \neq B^{rt_{i'j}}$.

Algorithm 4 *Verify*$(gpk, m, \sigma, RL_j, j)$

1: **Signature Check:**
2: Check that $B, J, K, L, T \in G_1$ and s_f, s_x, s_a, s_b, s_1, …, $s_\lambda \in \mathbb{Z}_p$.
3: Compute $R_1' = B^{s_f} J^{-c}$, $R_2' = B^{s_x} K^{-c}$, $R_4' = K^{s_a} B^{-s_b}$ and $R_3' = e(T, g_2)^{-s_x} T_2^{s_f} T_3^{s_b} T_4^{s_a} T_1^c e(T, w)^{-c}$.
4: Check that $c = H(gpk||B||J||K||L||T||R_1'||R_2'||R_3'||R_4'||j||m)$.
5: $\forall l = 1 \ldots \lambda$, let b_l be the lth bit of d. Compute $V_l' = B^{s_l} K^{b_l}$ and $W_l' = (B^{1-b_l} L^{b_l})^{h_j^{s_l}}$.
6: Check that $d = H(c'||(V_l', W_l')_{l=1\ldots\lambda})$.
7: **Revocation Check:**
8: Check that $\forall rt_{ij} \in RL_j, L \neq B^{rt_{ij}}$.
9: Output **valid** if all checks succeed. Otherwise output **invalid**.

The *Open* algorithm is described in Algorithm 5. This follows the usual "implicit tracing algorithm" described in [6]: the GM does a Revocation Check for the signature he would like to open (at period j), successively with every revocation list of the form $RL_j^i = \{rt_{ij}\}$, for each member M_i. When the check fails, it means that the signer is the one associated to the token in use.

Algorithm 5 *Open*$(gpk, m, \sigma, j, h_j, \{tk_i\}_i)$

1: $\forall i$, compute $rt_{ij} = h_j^{tk_i}$ and set $RL_j^i = \{rt_{ij}\}$.
2: $\forall i$, do the Revocation Check on σ using RL_j^i. When it fails (*i.e.* when the equality holds), break and return (tk_i, id_i).
3: If all checks succeed, return $\perp$.

5 An extractor for the [13] signature scheme

In [13], the *Sign* algorithm was slightly different. Of course, there was no proof of knowledge of a double logarithm. What is important to notice is that there was no R_4 value. The full algorithm is described in Algorithm 6. We do not re-write the *Verify* algorithm, it is easily extractable from Algorithm 4 by removing the corresponding elements.

Algorithm 6 The original *Sign*(gpk, sk_i, cre_i, m) from [13]

1: Choose $B \in_R G_1$ and compute $J = B^{f_i}$ and $K = B^{x_i}$.
2: Choose $a \in_R \mathbb{Z}_p$, compute $b = ax_i$ and $T = A_i \hat{g}_1^a$.
3: Choose $r_f, r_x, r_a, r_b \in_R \mathbb{Z}_p$.
4: Compute $R_3 = e(T, g_2)^{-r_x} T_2^{r_f} T_3^{r_b} T_4^{r_a}$, $R_1 = B^{r_f}$, $R_2 = B^{r_x}$
5: Compute $c = H(gpk||B||J||K||T||R_1||R_2||R_3||m)$.
6: Compute $s_f = r_f + cf_i$, $s_x = r_x + cx_i$, $s_a = r_a + ca$ and $s_b = r_b + cb$.
7: Output: $\sigma = (B, J, K, L, T, c, s_f, s_x, s_a, s_b)$.

We found out that it was possible to send s_a and s_b without following the original algorithm such that the signature is successfully checked. But we did not find a way to break the *Traceability* of the system. However, we think there is something missing in the proof of traceability in [13], that is quite summary and does not explicitly gives an extractor. This is why we add the R_4 part in our algorithms. Notice that adding R_4 does not change the signature size but only adds one multi-exponentiation in both *Sign* and *Verify* algorithms. Let us now prove that we can obtain an extractor when using the *Sign* procedure from Algorithm 3.

Theorem 1. *There exists an extractor for the group signature scheme as defined in Section 4, that extracts a valid key* (x, f, A) *from a convincing signer.*

Proof. We follow the usual methodology (see [6] for instance). We suppose that an extractor can rewind the protocol. We only consider the "non-backward-unlinkable" part of the signature, *i.e.* we remove L and the proof of knowledge of the double logarithm. The prover sends B, J, K, T, R_1, R_2, R_3, R_4. To a challenge value c, the prover responds with s_f, s_x, s_a, s_b. To another challenge value $c' \neq c$, the prover responds with s'_f, s'_x, s'_a, s'_b. As the prover is convincing,

all verification equations from Algorithm 4 hold. We denote $\Delta c = c - c'$, $\Delta s_f = s_f - s'_f$, $\Delta s_x = s_x - s'_x$, $\Delta s_a = s_a - s'_a$ and $\Delta s_b = s_b - s'_b$.

Now consider the R'_2 equation. Dividing the two instances gives us $B^{\Delta s_x} = K^{\Delta s_c}$. Let $\tilde{x} = \Delta s_x / \Delta c$. We obtain $B^{\tilde{x}} = K$. Let also $\tilde{f} = \Delta s_f / \Delta c$, $\tilde{a} = \Delta s_a / \Delta c$ and $\tilde{b} = \Delta s_b / \Delta c$

Now consider the R'_4 equation. We obtain $K^{\Delta s_a} = B^{\Delta s_b}$. With the previous result, we have $B^{\tilde{x}\Delta s_a} = B^{\Delta s_b}$ then $\tilde{x}\Delta s_a = \Delta s_b$ or $\tilde{x}\tilde{a} = \tilde{b}$.

Finally consider the R'_3 equation. We have

$$e(T, g_2)^{\Delta s_x} e(T, w)^{\Delta c} T_3^{-\Delta s_b} T_4^{-\Delta s_a} = T_2^{\Delta s_f} T_1^{\Delta c}$$

then

$$e(T, w g_2^{\tilde{x}}) e(\hat{g}_1, g_2)^{\tilde{b}} e(\hat{g}_1, w)^{\tilde{a}} = e(g_1 \tilde{g}_1^{\tilde{f}}, g_2).$$

Using the equality $\tilde{b} = \tilde{a}\tilde{x}$, we obtain

$$e(T \hat{g}_1^{-\tilde{a}}, w g_2^{\tilde{x}}) = e(g_1 \tilde{g}_1^{\tilde{f}}, g_2).$$

Set $\tilde{A} = T \hat{g}_1^{-\tilde{a}}$, we have

$$e(\tilde{A}, w g_2^{\tilde{x}}) = e(g_1 \tilde{g}_1^{\tilde{f}}, g_2)$$

and $(\tilde{x}, \tilde{f}, \tilde{A})$ is a valid key.

6 Security Proofs

In the Random Oracle Model (ROM), our VLR group signature scheme satisfies *Correctness*, *Backward Unlinkability* (under the modified DDH assumption), *Traceability* (under the SDH assumption) and *Exculpability* (under the DL assumption).

The *Correctness* is straightforward. The proofs for the other security properties are to be found in the following subsections.

6.1 Backward Unlinkability

Proposition 1. *Under the ROM and the hardness of modified DDH problem, the scheme described in Section 4 achieves Backward Unlinkability.*

Proof. Let us assume that there is an adversary $\mathcal{A}$ that succeeds with a non-negligible probability to break the *Backward Unlinkability* game. We describe here how to build a polynomial-time algorithm $\mathcal{B}$, that solves the modified-DDH problem in G_1 with a non-negligible probability.

$\mathcal{B}$ is given as input a modified-DDH tuple $(U, V = U^a, W = U^b, Z, u, v = u^a)$, where $U \in_R G_1$, $a, b \in_R \mathbb{Z}_p^*$, $Z \in G_1$ and $u \in_R \mathbb{Z}_p$. He decides if $Z = U^{ab}$ by interacting with $\mathcal{A}$ as follows:

1. Setup: $\mathcal{B}$ runs the *KeyGen* algorithm. The h_j's are not chosen completely randomly: $\forall j \in \{1, \ldots, T\}$, $\mathcal{B}$ chooses $r_j \in_R \mathbb{Z}_p$ and sets $h_j = u^{r_j}$.
   $\mathcal{B}$ obtains *gpk* and *ik*. *gpk* is sent to $\mathcal{A}$.

2. Hash Queries: We model the hash function H as a random oracle. When we do not specify that it should act otherwise, $\mathcal{B}$ chooses a random element of $\mathbb{Z}_p$, while ensuring consistency.
3. Join Queries: We assume there are n Join Queries[6]. $\mathcal{B}$ picks $i^* \in_R \{1, \ldots, n\}$. When $\mathcal{A}$ requests a new member for the ith time:
   - If $i = i^*$, $\mathcal{B}$ chooses $f \in_R \mathbb{Z}_p$ and sets $F = h_1^f$. $\mathcal{B}$ will act as this user had the credential $x = log_u v (= a)$ without knowing neither it nor the corresponding A. The revocation tokens for user M_{i^*} will be $rt_{i^*j} = v^{r_j}, \forall j \in \{1, \ldots, T\}$.
   - If $i \neq i^*$, $\mathcal{B}$ runs the *Join* protocol, playing both roles of the member and the issuer. The revocation tokens are also unchanged: $\forall j, rt_{ij} = h_j^{x_i}$.
4. Revocation Queries: If the requested i^{th} member and the actual period j are such that $i = i^*$ and $j = j^*$, $\mathcal{B}$ outputs a random guess $\beta \in_R \{0, 1\}$ and aborts the protocol. Otherwise, $\mathcal{B}$ outputs rt_{ij}.
5. Corruption Queries: If the requested member is M_{i^*}, $\mathcal{B}$ outputs a random guess $\beta \in_R \{0, 1\}$ and aborts the protocol. Otherwise, it gives the corresponding keys x, A, f, F to $\mathcal{A}$.
6. Signing Queries: If the requested signer M_i is not M_{i^*}, $\mathcal{B}$ answers to the query using the keys for M_i as usual.
   If $i = i^*$,
   - $\mathcal{B}$ chooses $r \in_R \mathbb{Z}_p$ and $T \in_R G_1$. He then computes $B = U^r$, $J = B^{f_{i^*}}$, $K = V^r$ and $L = B^{rt_{i^*j}} = U^{r.v^{r_j}}$.
   - $\mathcal{B}$ chooses c, d, s_f, s_x, s_a, s_b, s_1, $\ldots$, $s_\lambda \in_R \mathbb{Z}_p$.
   - $\mathcal{B}$ computes $R_1 = B^{s_f} J^{-c}$, $R_2 = B^{s_x} K^{-c}$,

     $$R_3 = e(T, g_2)^{-s_x} T_2^{s_f} T_3^{s_b} T_4^{s_a} T_1^c e(T, w)^{-c}$$

     and $R_4 = K^{s_a}\ B^{-s_b}$.
   - $\mathcal{B}$ patches the oracle by setting

     $$c = H(gpk||B||J||K||L||T||R_1||R_2||R_3||R_4||j||m).$$

     If this has been queried before, $\mathcal{B}$ outputs a random guess $\beta \in_R \{0, 1\}$ and aborts the protocol.
   - $\forall l = 1 \ldots \lambda$, let b_l be the lth bit of d. $\mathcal{B}$ computes $V_l = B^{s_l} K^{b_l}$ and $W_l = (B^{1-b_l} L^{b_l})^{h_j^{s_l}}$.
   - $\mathcal{B}$ patches the oracle by setting $d = H(c \text{——} (V_l, W_l)_{l=1\ldots\lambda})$. If this has been queried before, $\mathcal{B}$ outputs a random guess $\beta \in_R \{0, 1\}$ and aborts the protocol.
   - $\mathcal{B}$ returns to $\mathcal{A}$ $\sigma = (B, J, K, L, T, c, d, s_f, s_x, s_a, s_b, s_1, \ldots, s_\lambda)$.
7. Challenge: $\mathcal{A}$ outputs a message m^*, the current period j^* and two unrevoked and uncorrupted members M_{i_0} and M_{i_1}. If $i_0 \neq i^*$ and $i_1 \neq i^*$, $\mathcal{B}$ outputs a random guess $\beta \in_R \{0, 1\}$ and aborts the protocol.
   Otherwise, $\mathcal{B}$ chooses $b \in \{0, 1\}$ such that $i_b = i^*$ and builds a signature as follows:

[6] n is the number of *Join* requests needed by $\mathcal{A}$, it is consequently polynomial in k

- $\mathcal{B}$ chooses $r \in_R \mathbb{Z}_p$ and $T \in_R G_1$. He then computes $B = W^r$, $J = B^{f_{i^*}}$, $K = Z^r$ and $L = B^{rt_{i^*j}} = W^{r.v^{r_j}}$.
- $\mathcal{B}$ chooses $c, d, s_f, s_x, s_a, s_b, s_1, \ldots, s_\lambda \in_R \mathbb{Z}_p$.
- $\mathcal{B}$ computes $R_1 = B^{s_f} J^{-c}$, $R_2 = B^{s_x} K^{-c}$,

$$R_3 = e(T, g_2)^{-s_x} T_2^{s_f} T_3^{s_b} T_4^{s_a} T_1^c e(T, w)^{-c}$$

  and $R_4 = K^{s_a}\ B^{-s_b}$.
- $\mathcal{B}$ patches the oracle by setting

$$c = H(gpk||B||J||K||L||T||R_1||R_2||R_3||R_4||j||m).$$

  If this has been queried before, $\mathcal{B}$ outputs a random guess $\beta \in_R \{0,1\}$ and aborts the protocol.
- For $l = 1 \ldots \lambda$, let b_l be the lth bit of d. $\mathcal{B}$ computes $V_l = B^{s_l} K^{b_l}$ and $W_l = (B^{1-b_l} L^{b_l})^{h_j^{s_l}}$.
- $\mathcal{B}$ patches the oracle by setting $d = H(c \text{——} (V_l, W_l)_{l=1\ldots\lambda})$. If this has been queried before, $\mathcal{B}$ outputs a random guess $\beta \in_R \{0,1\}$ and aborts the protocol.
- $\mathcal{B}$ returns to $\mathcal{A}$ $\sigma^* = (B, J, K, L, T, c, d, s_f, s_x, s_a, s_b, s_1, \ldots, s_\lambda)$.

8. Restricted Queries: $\mathcal{A}$ and $\mathcal{B}$ interact in the same way as before the challenge, while respecting the restrictions.
9. Output: $\mathcal{A}$ outputs his guess $b' \in \{0,1\}$. If $b = b'$, $\mathcal{B}$ outputs 1, meaning that $Z = U^{ab}$, else $\mathcal{B}$ outputs 0.

If $\mathcal{B}$ never aborts:

- If $Z = U^{ab}$, then $\mathcal{B}$ simulates the game perfectly, *i.e.* the challenge signature σ^* is a well-simulated signature for user M_{i^*}. In this case, $\mathcal{A}$ outputs the good result with probability $1/2 + \epsilon$, and so does $\mathcal{B}$ for his modified-DDH instance.
- If $Z \neq U^{ab}$, then σ^* simulates a signature using a credential $x = log_W(Z)$, which is, except with negligible probability, different from the credentials of members M_{i_0} and M_{i_1}. In this case, $\mathcal{A}$ answers with probability $1/2$ $\mathcal{B}$'s choice or the other identity. $\mathcal{A}$ outputs the good result with probability $1/2$.

If $\mathcal{B}$ aborts, his output is random and he obtains the good result with probability $1/2$.

Let **abort** denote the event that $\mathcal{B}$ aborts. Let ϵ be the the probability of success of $\mathcal{A}$ against the *Backward Unlinkability* game. With the previous remarks, we have $\Pr(\overline{\mathbf{abort}})\epsilon$ as the advantage of $\mathcal{B}$ against modified-DDH. Let $\alpha \in \{0,1\}$ denote whether Z is random ($\alpha = 0$) or equal to U^{ab} ($\alpha = 1$) and β denote the guess of $\mathcal{B}$: $Adv_{\mathcal{B}} = \left|\Pr(\beta = 0|\alpha = 1) - \Pr(\beta = 0|\alpha = 0)\right| = \left|1 - \Pr(\beta = 1|\alpha = 1) - \Pr(\beta = 0|\alpha = 0)\right| = \left|1 - \Pr(\mathbf{abort})\Pr(\beta = 1|\mathbf{abort} \wedge \alpha = 1) - \Pr(\overline{\mathbf{abort}})\Pr(\beta = 1|\overline{\mathbf{abort}} \wedge \alpha = 1) - \Pr(\mathbf{abort})\Pr(\beta = 0|\mathbf{abort} \wedge \alpha = 1) - \Pr(\overline{\mathbf{abort}})\Pr(\beta = 0|\overline{\mathbf{abort}} \wedge \alpha = 1)\right| = \left|1 - \Pr(\mathbf{abort})\frac{1}{2} - \Pr(\overline{\mathbf{abort}})(\frac{1}{2} + \epsilon) - \Pr(\mathbf{abort})\frac{1}{2} - \Pr(\overline{\mathbf{abort}})\frac{1}{2}\right| = \Pr(\overline{\mathbf{abort}})\epsilon$.

Now let us evaluate ϵ. We can neglect the cases where the same hash queries are asked. Consequently, we neglect the abortions of the *Sign* phase. The protocol aborts only in the *Corruption*, *Revocation* and *Challenge* phases. The probability that it does not abort is the probability that i^* and j^* are chosen for the challenge, providing M_{i^*} has not been either corrupted or revoked at time j^*. Let q_C denote the number of revocation queries and q_R the maximum number of revocation queries for one period. The probability that M_{i^*} is neither corrupted nor revoked is $\geq 1 - \frac{q_R+q_C}{n}$. The probability that j^* is chosen is $1/T$. If i^* is available for the challenge choice, he has a probability $\geq 2/n$ to be chosen. Consequently, we have $Adv_{\mathcal{B}} \geq (1 - \frac{q_R+q_C}{n})\frac{2}{nT}\epsilon$.

Thus, as n is polynomial in k, the advantage of $\mathcal{B}$ against modified DDH in G_1 is non-negligible and, consequently, our scheme satisfies *Backward Unlinkability* under the modified DDH assumption in G_1.

6.2 Traceability

Proposition 2. *Under the ROM and the SDH assumption in (G_1, G_2, G_T), the scheme defined in Section 4 achieves Traceability.*

Proof. Let us assume that there is an adversary $\mathcal{A}$ that succeeds with a non-negligible probability to break the *Traceability* game. We describe here how to build a polynomial-time algorithm $\mathcal{B}$, that solves the q-SDH problem with a non-negligible probability.

$\mathcal{B}$ is given as input a n-SDH instance, obtained from a triple (g_1, g_2, w). He can then compute $n-1$ SDH pairs (B_k, x_k) using techniques from [5]. He builds a new SDH pair by interacting with $\mathcal{A}$ as follows:

1. Setup: $\mathcal{B}$ runs the *KeyGen* algorithm with one modification. He picks $a, b, x \in_R \mathbb{Z}_p^*$ and sets
$$\tilde{g}_1 = \psi(((w g_2^x)^b g_2^{-1})^{1/a}) = g_1^{((\gamma+x)b-1)/a}.$$
Notice that $\mathcal{B}$ knows w but not γ. $\mathcal{B}$ obtains $gpk = (G_1, G_2, G_T, p, e, g_1, \tilde{g}_1, \hat{g}_1, g_2, w, H, T_1, T_2, T_3, T_4)$ and sends it to $\mathcal{A}$.
2. Hash Queries: We model the hash function H as a random oracle. When we do not specify that it should act otherwise, $\mathcal{B}$ chooses a random element of $\mathbb{Z}_p$, while ensuring consistency.
3. Join Queries: Let us remind that there are 2 cases of join queries: in the first case, $\mathcal{B}$ runs the *Join* algorithm alone, in the second case $\mathcal{A}$ and $\mathcal{B}$ interact. We assume there are n Join Queries. $\mathcal{B}$ picks $i^* \in_R \{1, \ldots, n\}$. We assume that the i^{*th} query is a query of the first type. When $\mathcal{A}$ requests a new member for the ith time:
   - If $i = i^*$, $\mathcal{B}$ sets $f_{i^*} = a$ and $cre_i = (g_1^b, x)$.
   - If $i \neq i^*$, depending on the cases, $\mathcal{B}$ receives f_i from $\mathcal{A}$ or chooses $f_i \in_R \mathbb{Z}_p$. $\mathcal{B}$ uses then one SDH pair (B_k, x_k) to compute cre_i: $x_i = x_k$ and $A_i = B_k^{1-f_i/a+f_i b(x-x_i)/a} g_1^{f_i b/a} = B_k^{1-f_i/a+f_i b(x+\gamma)/a} = (g_1 \tilde{g}_1^{f_i})^{1/(\gamma+x_i)}$.

4. Corruption Queries: At period j, $\mathcal{B}$ gives the secret key sk_i and the credential cre_i of the requested member M_i. The revocation tokens for M_i for periods $j' \geq j$ are added to the $RL_{j'}$'s.
5. Signing Queries: Given a signer identity, a message m to be signed and a period j, $\mathcal{B}$ computes the signature σ using the *Sign* algorithm and sends it to $\mathcal{A}$.
6. Output: $\mathcal{A}$ outputs a signer's identity, a message m and a signature σ.

We can treat two types of forgeries :

1. $\mathcal{A}$ outputs a signature with secret key A^* different from all A_i^*'s.
2. $\mathcal{A}$ outputs a signature with the secret key of an uncorrupted member. $\mathcal{B}$ obtains an advantage against his SDH instance only if $\mathcal{A}$ signs using M_{i^*}'s keys. This happens with probability $1/n$ (non-negligible).

In both cases, $\mathcal{B}$ can rewind the framework to obtain two forged signatures on the same message m at the same interval j using the sames B, K, T, R_1, R_2.

Using the Forking Lemma [20] and our extractor described in Section 5, $\mathcal{B}$ is able to compute a new SDH pair with non-negligible probability in polynomial time.

Thus, the scheme satisfies *Traceability* under the q-SDH assumption in (G_1, G_2, G_T).

6.3 Exculpability

Proposition 3. *Under the ROM and the DL assumption, the scheme described in Section 4 achieves Exculpability.*

Proof. Let us assume that there is an adversary $\mathcal{A}$ that succeeds with a non-negligible probability to break the *Exculpability* game. We describe here how to build a polynomial-time algorithm $\mathcal{B}$, that solves the DL problem with a non-negligible probability.

$\mathcal{B}$ is given as input a DL instance: (g, h). He finds $log_g h$ by interacting with $\mathcal{A}$ as follows:

1. Setup: $\mathcal{B}$ runs the *KeyGen* algorithm with one modification. He does not choose $\tilde{g}_1$ randomly but sets $\tilde{g}_1 = g$. The rest is not modified. $\mathcal{B}$ sends gpk and ik to $\mathcal{A}$, stores gpk and sets revocation lists.
2. Join Queries: We assume there are n Join Queries. $\mathcal{B}$ picks $i^* \in_R \{1, \ldots, n\}$. When $\mathcal{A}$ requests the creation of new member, $\mathcal{A}$ and $\mathcal{B}$ play the *Join algorithm*, $\mathcal{A}$ as the GM and $\mathcal{B}$ as the joining member. There is only one restriction: for the i^{*th} query, $\mathcal{B}$ does not pick f_i^* randomly but sets $id_{i^*} = F_{i^*} = h$ and simulates the proof of knowledge of $log_{\tilde{g}_1} F_{i^*}$. So $sk_{i^*} = log_g h$ but $\mathcal{A}$ does not know its value.
3. Corruption Queries: At period j, $\mathcal{B}$ gives the secret key sk_i and the credential cre_i of the requested member M_i. The revocation tokens for M_i for periods $j' \geq j$ are added to the $RL_{j'}$'s.
   If the requested member is M_{i^*}, $\mathcal{B}$ aborts the protocol.

4. Signing Queries: If $\mathcal{A}$ requests the signature of a member $M_i \neq M_{i^*}$, $\mathcal{B}$ answers using the *Sign* algorithm normally. If the requested member is M_{i^*}, $\mathcal{B}$ runs *Sign* but does not choose B randomly, he chooses $t \in_R \mathbb{Z}_p$ and sets $B = g^t$ and $J = h^t$. K, L and T are computed as usual. The first proof of knowledge is simulated because $\mathcal{B}$ does not know f_{i^*} and the second proof of knowledge, concerning K and L is done normally.
5. Output: $\mathcal{A}$ outputs a message m, a signature $\sigma = (B, J, K, L, T, \Pi_1, \Pi_2)$, the current interval j, the revocation list RL_j and a tracing key pair (id_i, tk_i).

$\mathcal{B}$ has an advantage against the DL instance if the latter key pair corresponds to user M_{i^*}. As i^* looks random for $\mathcal{A}$, it is chosen with probability at least $1/n$ (consequently with non-negligible property).

If we assume that $\mathcal{A}$ wins the *Exculpability* game, we can state that $J = B^{f_{i^*}}$, since this statement is indisputable.

Using the Forking Lemma [20], one can then extract f_{i^*} with non-negligible probability. Consequently, $\mathcal{B}$ finds $log_h g$, the solution for his DL instance, with non-negligible probability in polynomial time.

Thus, our scheme is *exculpable* under the Discrete Logarithm assumption in G_1.

7 Efficiency

In this section, we analyse our scheme in terms of signature size and computation cost. We then draw a comparison of the performances of our scheme and some other VLR schemes.

7.1 Analysis of the Proposed Scheme

We compare here our proposal with the patched [13] scheme. Notice that we add in the signature 2λ elements (V_l, W_l) of $\mathbb{Z}_p$ and one element L of G_1. Chen and Li proposed to use 256-bit Barreto-Naehrig curves [3]. In this context, each element of G_1 can be represented using 257 bits. A [13] signature using these parameters is 2308-bit long. A signature from our scheme using a security parameter $\lambda = 80$ is 23301-bit long, *i.e.* about ten times bigger. Concerning computation times, our modification requires $2\lambda+1$ additional exponentiations for the signing part and 2λ additional exponentiations for the verifying part. Note that, despite this overhead, one important property of our solution is that revocation check remains as fast as in the original scheme. This is summed up in Table 2. (me stands for multi-exponentiations in G_1, ME for multi-exponentiations in G_T and P for pairings. The patched [13] scheme is the [13] scheme modified to obtain an extractor, *i.e.* our scheme without the proof of knowledge of a double logarithm.)

Scheme	Cost of *Sign*	Cost of *Verify*
patched [13]	6 me + 1 ME	$(4 + \|RL\|)$ me + 1 ME + 1 P
Our scheme	$(7 + 2\lambda)$ me + 1 ME	$(4+ \|RL_j\| + 2\ \lambda)$ me+ 1 ME + 1 P

Table 2. Computational costs for [13] and our scheme

7.2 Comparison with Existing Works

We now compare the additional cost for Backward Unlinkability for the [13] scheme and for a pairing-based scheme. We use as an example the [6] and the [18] schemes, the latter being a modification of the Boneh-Shacham scheme enabling Backward Unlinkability. We implemented these algorithms on a PC with a 2.93 GHz Intel®Core^{TM}2 Duo processor. The implementation uses the C++ programming language and the NTL library [21]. The order of the groups used in the implementation is a 160-bit integer. We compute pairings using an optimization technique from [24]. One can find in Table 3 our results for the schemes [6] and our correction of [13] without Backward Unlinkability (denoted by patched CL). We give the computation times for the *Sign* algorithm, for the constant part of the *Verify* algorithm and, finally, the time needed for each additional revoked user. Our results imply that computing a pairing is about four times longer than computing an exponentiation, which confirms the improvement brought by exponentiations in terms of efficiency.

Scheme	BS [6]	patched CL
Signature	1000	450
Verification	1170	400
Rev. Check (/rev.)	180	45

Table 3. Computation times for the schemes without BU, in ms.

In Table 4, we describe the additional time needed to add Backward Unlinkability to these schemes. The operations in the *Revocation Check* part have the same cost, that is why they are not mentioned here. We denote by CL-BU$_\lambda$ our scheme with a security parameter λ. We can see that, for pairing based schemes ([6, 18]), BU is essentially for free. In our scheme, it requires about 100 more milliseconds per security level, which is coherent with the theoretical costs of Table 2.

We also show why, despite the additional cost due to the security parameter of the proof of knowledge, our scheme becomes quickly more efficient than

Scheme with/ without BU	NF [18]/BS [6]	CL-BU$_{80}$/CL	CL-BU$_{128}$/CL
Signature	80 ms	8 s	13 s
Verification	40 ms	8 s	13 s

Table 4. Additional time for Backward Unlinkability.

a pairing-based scheme with BU. In Table 5, we show the time needed by the *Verify* algorithm for the [18] scheme and for our scheme, using different security parameters. We can see that there is a threshold value for the number of revoked members from which our scheme is more efficient. Our scheme is the most adapted for large groups (from a few dozens of users).

For instance, we remark that the overall time for signing and verifying when there are 1000 revoked members is divided by 3 for our scheme CL-BU$_{80}$ compared to the [18] scheme.

Revoked members	NF [18]	CL-BU$_{80}$	CL-BU$_{128}$
10	3 s	9 s	14 s
50	10 s	10.5 s	15.5 s
100	19 s	13 s	18 s
1000	3 min	53 s	58 s

Table 5. Computational time for the *Verify* algorithm, depending on the number of revoked members.

8 Conclusion

We describe in this paper a technique to add Backward Unlinkability to VLR group signature schemes using exponentiations in the Revocation Check.

By applying our technique to [13], that we moreover modified to give a full security proof for traceability, we obtain the most efficient VLR scheme enabling Backward Unlinkability. It also satisfies Exculpability and full proofs are given.

Our technique does not break the linearity (in the number of revoked members) of the Revocation Check. It is still an open problem to prove if we can build a VLR scheme with a sublinear Revocation Check or not.

Acknowledgment

The authors thank Hervé Chabanne for his valuable comments that helped to improve the quality of the paper.

References

1. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In M. Bellare, editor, *CRYPTO*, volume 1880 of *LNCS*, pages 255–270. Springer, 2000.
2. G. Ateniese, D. X. Song, and G. Tsudik. Quasi-efficient revocation in group signatures. In M. Blaze, editor, *Financial Cryptography*, volume 2357 of *LNCS*, pages 183–197. Springer, 2002.
3. P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. E. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *LNCS*, pages 319–331. Springer, 2005.
4. M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, pages 136–153, 2005.
5. D. Boneh and X. Boyen. Short signatures without random oracles. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *LNCS*, pages 56–73. Springer, 2004.
6. D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In V. Atluri, B. Pfitzmann, and P. D. McDaniel, editors, *ACM Conference on Computer and Communications Security*, pages 168–177. ACM, 2004.
7. E. Brickell and J. Li. A pairing-based daa scheme further reducing tpm resources. In A. Acquisti, S. W. Smith, and A.-R. Sadeghi, editors, *TRUST*, volume 6101 of *LNCS*, pages 181–195. Springer, 2010.
8. E. F. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In V. Atluri, B. Pfitzmann, and P. D. McDaniel, editors, *ACM Conference on Computer and Communications Security*, pages 132–145. ACM, 2004.
9. J. Bringer, H. Chabanne, D. Pointcheval, and S. Zimmer. An application of the Boneh and Shacham group signature scheme to biometric authentication. In K. Matsuura and E. Fujisaki, editors, *IWSEC*, volume 5312 of *LNCS*, pages 219–230. Springer, 2008.
10. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *CRYPTO*, volume 2442 of *LNCS*, pages 61–76. Springer, 2002.
11. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In B. S. K. Jr., editor, *CRYPTO*, volume 1294 of *LNCS*, pages 410–424. Springer, 1997.
12. D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.
13. L. Chen and J. Li. VLR group signatures with indisputable exculpability and efficient revocation. In *PASSAT*, 2010.
14. L. Ibraimi, S. Nikova, P. Hartel, and W. Jonker. An identity-based group signature with membership revocation in the standard model. CTIT technical report series. University of Twente, 2010.
15. A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *LNCS*, pages 571–589. Springer, 2004.
16. B. Libert and D. Vergnaud. Group signatures with verifier-local revocation and backward unlinkability in the standard model. In J. A. Garay, A. Miyaji, and A. Otsuka, editors, *CANS*, volume 5888 of *LNCS*, pages 498–517. Springer, 2009.
17. T. Nakanishi and N. Funabiki. Verifier-local revocation group signature schemes with backward unlinkability from bilinear maps. In B. K. Roy, editor, *ASIACRYPT*, volume 3788 of *LNCS*, pages 533–548. Springer, 2005.

18. T. Nakanishi and N. Funabiki. A short verifier-local revocation group signature scheme with backward unlinkability. In H. Yoshiura, K. Sakurai, K. Rannenberg, Y. Murayama, and S. ichi Kawamura, editors, *IWSEC*, volume 4266 of *LNCS*, pages 17–32. Springer, 2006.
19. T. Nakanishi, A. Sudarsono, Y. Sakemi, Y. Nogami, and N. Funabiki. A group signature scheme with efficient verifier-local revocation check. In *SCIS*, 2009.
20. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.
21. V. Shoup. Number theory library. `http://www.shoup.net/ntl`.
22. D. X. Song. Practical forward secure group signature schemes. In *ACM Conference on Computer and Communications Security*, pages 225–234, 2001.
23. M. Stadler. Publicly verifiable secret sharing. In *EUROCRYPT*, pages 190–199, 1996.
24. M. Stogbauer. Efficient algorithms for pairing-based cryptosystems. Master's thesis, Darmstadt University of Technology, 2004.
25. A. Studer, E. Shi, F. Bai, and A. Perrig. Tacking together efficient authentication, revocation, and privacy in vanets. Technical report, Carnegie Mellon CyLab, 2008.
26. A. Sudarsono, T. Nakanishi, Y. Sakemi, Y. Nogami, and N. Funabiki. An implementation of a group signature scheme with efficient verifier-local revocation check. In *ISEC*, pages 37–42, 2009.
27. L. Wei and J. Liu. Shorter verifier-local revocation group signature with backward unlinkability. In M. Joye, A. Miyaji, and A. Otsuka, editors, *Pairing*, volume 6487 of *LNCS*, pages 136–146. Springer, 2010.
28. T. Yoshida and K. Okada. Simple and efficient group signature scheme assuming tamperproof devices. In K. Matsuura and E. Fujisaki, editors, *IWSEC*, volume 5312 of *LNCS*, pages 83–99. Springer, 2008.
29. S. Zhou and D. Lin. Shorter verifier-local revocation group signatures from bilinear maps. In D. Pointcheval, Y. Mu, and K. Chen, editors, *CANS*, volume 4301 of *LNCS*, pages 126–143. Springer, 2006.