Chapter 31
Fully Dynamic Group Signature Scheme with Member Registration and Verifier-Local Revocation

Maharage Nisansala Sevwandi Perera and Takeshi Koshiba

Abstract Since Bellare et al. (EUROCRYPT 2003) proposed a security model for group signature schemes, almost all the securities of group signature schemes have been discussed in their model (the BMW03 model). While the BMW03 model is for static groups, Bellare et al. in 2005 considered the case of dynamic group signature schemes and provided a solution to cope with dynamic groups. However, their scheme does not serve member revocation, serves only member registration. In this paper, we incorporate a member revocation mechanism into a group signature scheme with member registration and construct a fully dynamic group signature, which supports verifier-local revocation (VLR) to manipulate member revocation. Moreover, we achieve the security of the proposed scheme with a restricted version of full anonymity to overcome the security complications that may arise due to member revocation.

Keywords Dynamic group signature · Verifier-local revocation · Almost-full anonymity

1 Introduction

The notion of group signature was first introduced by Chaum and van Heyst [12] in 1991. Each member has a private signing key and a corresponding public key. The private signing key is used to generate signatures on messages while the public key is used as a public verification key by verifiers to authenticate the signatures. Group signatures allow group members to sign anonymously on behalf of the group (*anonymity*). Only the authorized person can reveal the identity of the member who signs (*traceability*).

M. N. S. Perera (✉)
Graduate School of Science and Engineering, Saitama University, Saitama, Japan
e-mail: perera.m.n.s.119@ms.saitama-u.ac.jp

T. Koshiba
Faculty of Education and Integrated Arts and Sciences, Waseda University, Tokyo, Japan
e-mail: tkoshiba@waseda.jp

D. Ghosh et al. (eds.), *Mathematics and Computing*, Springer Proceedings in Mathematics & Statistics 253, https://doi.org/10.1007/978-981-13-2095-8_31

Besides the naive security notions (*anonymity* and *traceability*) for group signatures, more security requirements like un-frameability, collusion resistance, and unforgeability are proposed. In 2003, Bellare et al. [2] suggested a formal security notion with *full anonymity* and *full traceability* to provide a stronger security for group signature schemes (the BMW03 model). This BMW03 model supports only for static groups, not for dynamic groups. Hence, it does not guarantee the security when group members can be flexibly reorganized.

In the setting of dynamic group signatures, neither the number of group members nor their keys should be fixed in the setup phase. Thus, a scheme should be able to register or revoke members anytime. In 2005, Bellare et al. [3] suggested a scheme by providing foundations for dynamic group signatures. The scheme in [3] helps to bridge the gap between the results in [2], and the previous works are done to deliver a dynamic group signature scheme. The dynamic groups are more complex than the static groups since they require many security concerns and deliver more issues to be focused. Schemes in [3] and [14] provide formal security definitions for dynamic group signatures to overcome those issues. Another scheme was suggested by Libert et. al. [15]. However, none of them are fully dynamic group signature schemes since they do not support member revocation. Recently, Bootle et. al. [7] suggested a security definition for fully dynamic group signature schemes and they have also provided some fixes for existing schemes. Hereafter, if a scheme supports both member registration and member revocation, we refer to it as fully dynamic, and if a scheme supports either member registration or revocation, we refer to it as dynamic.

The member revocation is an essential requirement in practice, and many researchers presented various approaches to manage member revocation in groups. One approach is replacing the group public key and the private signing keys with new keys for all existing members when a member is revoked. Since this requires to update all the existing members and the verifiers, it is not the best solution, especially not suitable for large groups. In 2001, Bresson et al. [8] provided a solution that requires signers to prove, at the time of signing, that their member certificates are not in the public revocation list. In 2002, Camenisch et al. [11] proposed a different approach, which is based on dynamic accumulators. It maps a set of values into a fixed-length string and permits efficient proofs of memberships. However, this approach requires existing members to keep track of the revoked users. Thus, it increases the workload of existing members. Moreover, schemes in [5, 10, 18] have taken some other revocation approache.

A different and simple revocation mechanism was suggested by Brickell [9], which was subsequently formalized by Boneh and Shacham [6]. This revocation mechanism is known as *Verifier-Local Revocation* (*VLR*). *VLR* allows the members to convince the verifiers that they are valid members, who are not revoked and eligible to sign on behalf of the group. Every member has a unique token, and when he is revoked, this token is added to a list called *Revocation List* (*RL*). Then, the group manager passes the latest *RL* to the verifiers. When a verifier needs to authenticate a signature, he checks the validity of the signer with the help of *RL*. Since the verifiers are smaller in number than the members, this mechanism is more convenient than any

others, especially for large groups. Moreover, this is advantageous to the previous approaches since it does not affect on existing members.

Our Contribution

This paper presents a fully dynamic group signature scheme that allows to both add and revoke members and a new security notion to overcome some security barriers.

First, we take the scheme in [3], which includes an interactive protocol, that allows new users to join the group at any time, and we incorporate with member revocation mechanism by adapting the methods in the scheme in [3] and suggesting new methods to manage member revocation with *VLR*.

Then, we suggest a method to generate member revocation tokens in our scheme. In general, any VLR scheme consists of a token system and those tokens are generated as a part of the secret signing key. Since our intention is to apply full anonymity which requires to provide all the secret signing keys to an adversary at the anonymity game, this method is not suitable for our scheme. If we generate revocation tokens using the signing keys of the members, the adversary can obtain the tokens of the challenged indices and win the anonymity game. Thus, to present a member's token, we use his personal secret key ($usk[i]$) and his verification key (pk_i). Nevertheless, pk_i is a public attribute, revealing pk_i does not show any other information about the member. Even though $usk[i]$ is a secret key, no one can generate any secret signing key by $usk[i]$. Besides, no one can create a group member token using the secret signing key, since the token is not a part of the secret signing key. Thus, it ensures the security of the scheme.

Moreover, we present a new security notion that is somewhat weaker than the full anonymity. *VLR* relies on a weaker security notion called *selfless-anonymity*. Even our intention is to apply full anonymity for our scheme, achieving *full anonymity* suggested in the BMW03 model for *VLR* is quite difficult. In case of the anonymity game (for the definition) between a challenger and a adversary, the BMW03 model passes all the secret keys to the adversary. But, we cannot allow the adversary to reveal all the secret keys since he can corrupt the anonymity of the scheme. If we allow the adversary to reveal all the users' personal private keys (*usk*), which we use to create tokens he can create any token, including the challenged users' tokens. Then, he can verify the challenging signature and return the correct user index of the challenged signature. Thus, we suggest a *new* restricted version of full anonymity (*almost-full anonymity*), which will not provide all the secret keys to the adversary to ensure the security of our scheme. It will allow the adversary to reveal any member's secret signing keys not the member's personal private keys.

2 Preliminaries

In this section, we describe notations used in the paper and the primitives with which we use to construct our scheme. Construction of dynamic group signature schemes use three building blocks: public-key encryption schemes secure against

chosen-ciphertext attack [13], digital signature schemes secure against chosen-message attack [1], and simulation-sound adaptive non-interactive zero-knowledge (NIZK) proofs for NP [17]. All the three primitives are based on trapdoor permutation.

2.1 Notation

We denote by λ the security parameter of the scheme and let $\mathbb{N} = \{1, 2, 3, \ldots\}$ be the set of *positive integers*. For any $k \geq 1 \in \mathbb{N}$, we denote by $[k]$ the set of integers $\{1, \ldots, k\}$. An empty string is denoted by ε. If s is a string, then $|s|$ denotes the length of the string and if $\mathcal{S}$ is a set then $|\mathcal{S}|$ denotes the size of the set. If $\mathcal{S}$ is a finite set, $b \xleftarrow{\$} \mathcal{S}$ denotes that b is chosen uniformly at random from $\mathcal{S}$. We denote experiments by **Exp**.

2.2 Digital Signature Schemes

A digital signature scheme $DS = (\mathsf{K}_s, \mathsf{Sig}, \mathsf{Vf})$ consists of three algorithms: key generation K_s, signing Sig, and verification Vf. The scheme DS should satisfy the standard notion of unforgeability under chosen-message attack.

For an adversary $\mathcal{A}$, consider an experiment $\mathbf{Exp}^{\textit{unforg-cma}}_{DS,\mathcal{A}}(\lambda)$. First, a pair of a public key and the corresponding secret key for the scheme DS is obtained by executing K_s with the security parameter λ as $(\mathbf{pk}, \mathbf{sk}) \xleftarrow{\$} \mathsf{K}_s(1^\lambda)$. Then, the public key $\mathbf{pk}$ is given to the adversary, and the adversary can access the signing oracle $\mathsf{Sig}(\mathbf{sk}, \cdot)$ for any number of messages. Finally, the forging adversary $\mathcal{A}$ outputs (m, σ). He wins if σ is a valid signature on the message m and m is not queried so far. We let $\mathbf{Adv}^{\textit{unforg-cma}}_{DS,\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\textit{unforg-cma}}_{DS,\mathcal{A}}(\lambda) = 1]$.

A digital signature scheme DS is secure against forgeries under chose message attack if $\mathbf{Adv}^{\textit{unforg-cma}}_{DS,\mathcal{A}}(\lambda)$ is negligible in λ for any polynomial-time adversary $\mathcal{A}$.

2.3 Encryption Scheme

An encryption scheme $E = (\mathsf{K}_e, \mathsf{Enc}, \mathsf{Dec})$ consists of three algorithms: key generation K_e, encryption Enc, and decryption Dec. The scheme E should satisfy the standard notion of indistinguishability under adaptive chosen-ciphertext attack.

For an adversary $\mathcal{A}$, consider an experiment $\mathbf{Exp}^{\textit{ind-cca-b}}_{E,\mathcal{A}}(\lambda)$. First, a pair of a public key and the corresponding secret key for the encryption scheme E is obtained by executing K_e with the security parameter λ and a randomness string r_e (where the length of r_e is bounded by some fixed polynomial $r(\lambda)$) as $(\mathbf{pk}, \mathbf{sk}) \xleftarrow{\$} \mathsf{K}_e(1^\lambda, r_e)$.

Let $\mathsf{LR}(m_0, m_1, b)$ a function which returns m_b for a bit b and messages m_0, m_1. We assume the adversary $\mathcal{A}$ never queries $\mathsf{Dec}(\mathbf{sk}, \cdot)$ on a ciphertext previously returned by $\mathsf{Enc}(\mathbf{pk}, \mathsf{LR}(\cdot, \cdot, b))$. We let $\mathbf{Adv}^{ind\text{-}cca}_{E,A}(\lambda) = |\Pr[\mathbf{Exp}^{ind\text{-}cca\text{-}1}_{E,\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Exp}^{ind\text{-}cca\text{-}0}_{E,\mathcal{A}}(\lambda) = 1]|$.

An encryption scheme E is IND-CCA secure if $\mathbf{Adv}^{ind\text{-}cca}_{E,\mathcal{A}}(\lambda)$ is negligible in λ for any polynomial-time adversary $\mathcal{A}$.

2.4 Simulation-Sound Non-interactive Zero-Knowledge Proof System

A two-party game between a prover and a verifier which needs to determine whether a given string belongs to a language or not is called an interactive system. The interactive system allows to exchange messages between the prover and the verifier. Besides, argument systems are like interactive proof systems, except they are required to be computationally infeasible for a prover to convince the verifier to accept inputs not in the language. Non-interactive proof systems are mono-directional [4]. The non-interactive proof systems allow a prover to convince a verifier about a truth statement while zero-knowledge ensures that the verifier learns nothing from the proof other than the truth of the statement. The non-interactive zero-knowledge proof system shows that without any interaction but using a common string computational zero-knowledge can be achieved. In a simulation-sound NIZK proof system, an adversary cannot prove any false statements even after seeing simulated proofs of arbitrary statements.

An NP-*relation over domain* Dom $\subseteq \{0,1\}^*$ is a subset ρ of $\{0,1\}^* \times \{0,1\}^*$. We say that x is a *theorem* and w is a *proof* of x if $(x, w) \in \rho$. The membership of $(x, w) \in \rho$ is decidable in time polynomial in the length of the first argument for all x in Dom.

We fix an NP relation ρ over Dom and take a pair of polynomial-time algorithms (P, V), where P is randomized, and V is deterministic. Both P and V have access to a *common reference string* R. The (P, V) is a non-interactive proof system for ρ over Dom if the following two conditions are satisfied for polynomials p and l.

- *Completeness*: $\forall\lambda \in \mathbb{N}, \forall(x, w) \in \rho$ with $|x| \leq l(\lambda)$ and $x \in Dom$:
  $\Pr[R \xleftarrow{\$} \{0,1\}^{p(\lambda)};\ \pi \xleftarrow{\$} P(1^\lambda, x, w, R) : V(1^\lambda, x, \pi, R) = 1] = 1.$
- *Soundness*: $\forall\lambda \in \mathbb{N}, \forall \hat{P}$ and $x \in Dom$ such that $x \notin L_\rho$:
  $\Pr[R \xleftarrow{\$} \{0,1\}^{p(\lambda)};\ \pi \xleftarrow{\$} \hat{P}(1^\lambda, x, R) : V(1^\lambda, x, \pi, R) = 1] \leq 2^{-\lambda}.$

3 Our Scheme

We construct our scheme based on the scheme in [3]. In the scheme in [3], they have taken a digital signature scheme $DS = (\mathsf{K}_s, \mathsf{Sig}, \mathsf{Vf})$ and a public-key encryption scheme $E = (\mathsf{K}_e, \mathsf{Enc}, \mathsf{Dec})$ as the building blocks to construct a group signature

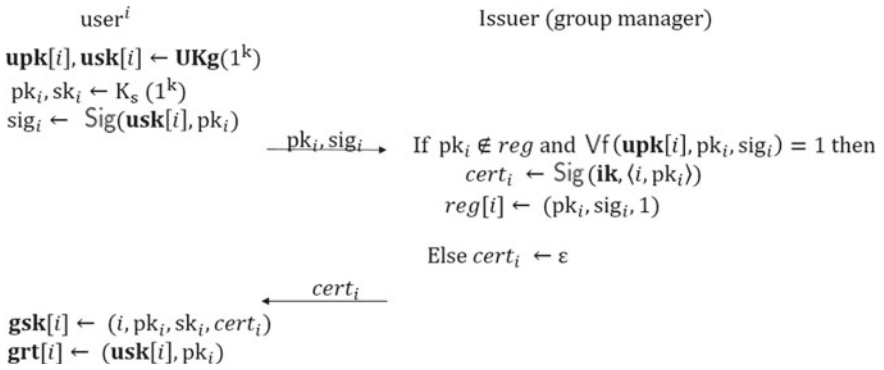

Fig. 1 Group-joining protocol

scheme *GS*. Moreover, they have used NIZK proof system to convince the verifier the validity of the signature. We also use above-mentioned primitives; *DS*, *E*, and NIZK to present a new scheme *FDGS* = (GKg, UKg, Join, Issue, Revoke, Sign, Verify, Open, Judge). GKg, UKg, and Judge are same as the scheme in [3]. We provide a new algorithm Revoke to revoke members, and we modify Join, Issue, Sign, Verify, and Open to be compatible with the revocation mechanism. We use *DS* for generating the group manager's keys and *E* for generating the opener's keys. Thus, our group public key **gpk** consists of the security parameter λ, public keys of *group manager* and *opener*, and two reference strings R_1, R_2 obtained for NIZK proof.

We describe our *group-joining protocol* which executes Join and Issue in Fig. 1, and we describe other algorithms of our scheme in Fig. 2.

3.1 Coping with VLR and Making the Scheme Secure

In general, *VLR* schemes satisfy a weaker security notion called *selfless-anonymity*, which does not provide any secret keys to the adversary. Even though our scheme supports *VLR* mechanism, we make our scheme more secure by using the techniques in [3] scheme and suggesting a new security notion called *almost-full anonymity*. Making VLR scheme fully anonymous is quite difficult since the full anonymity requires to provide all the secret keys to the adversary and providing tokens to the adversary makes the scheme insecure. The adversary can execute Verify with the tokens of the challenged indices and win the game easily. Thus, we consider a new security notion called *almost-full anonymity* which will not provide tokens to the adversary, which is a restricted version of the full anonymity.

Moreover, any *VLR* scheme has an associated tracing mechanism called *implicit tracing algorithm* to trace signers. The implicit tracing algorithm requires to run Verify linear times in the number of group members. Compare to the *explicit tracing*

GKg$(1^\lambda) \to (\mathbf{gpk}, \mathbf{ok}, \mathbf{ik})$

$R_1 \xleftarrow{\$} \{0,1\}^{P1(\lambda)}$.
$R_2 \xleftarrow{\$} \{0,1\}^{P2(\lambda)}$.
$r_e \xleftarrow{\$} \{0,1\}^{r(\lambda)}$.
$(opk, osk) \leftarrow K_e(1^\lambda; r_e)$.
$(gmpk, gmsk) \xleftarrow{\$} K_s(1^\lambda)$.
$\mathbf{gpk} \leftarrow (1^\lambda, R_1, R_2, opk, gmpk)$.
$\mathbf{ok} \leftarrow (osk, r_e)$.
$\mathbf{ik} \leftarrow gmsk$.
Return ($\mathbf{gpk}$,$\mathbf{ok}$,$\mathbf{ik}$).

Sign$(\mathbf{gpk}, \mathbf{gsk}[i], \mathbf{grt}[i], m) \to (\sigma)$

Parse $\mathbf{gpk}$ as $(1^\lambda, R_1, R_2, opk, gmpk)$.
Parse $\mathbf{gsk}[i]$ as $(i, pk_i, sk_i, cert_i)$.
$s \leftarrow \mathsf{Sig}(sk_i, m); r \xleftarrow{\$} \{0,1\}^\lambda$.
$C \leftarrow \mathsf{Enc}(opk, \langle i, pk_i, cert_i, s\rangle; r)$.
$\pi_1 \xleftarrow{\$} P_1(1^\lambda, (opk, gmpk, m, C),$
$(i, pk_i, cert_i, s, r), R_1)$.
$\sigma \leftarrow (C, \pi_1, \mathbf{grt}[i])$.
Return σ.

Open$(\mathbf{gpk}, \mathbf{ok}, reg, m, \sigma) \to (i, \tau, st)$

Parse $\mathbf{gpk}$ as $(1^\lambda, R_1, R_2, opk, gmpk)$.
Parse $\mathbf{ok}$ as (osk, r_e).
Parse σ as $(C, \pi_1, \mathbf{grt}[i])$.
$M \leftarrow Dec(osk, C)$.
Parse M as $\langle i, pk, cert, s\rangle$.
If $reg[i] \neq \varepsilon$ then
Parse $reg[i]$ as $(pk_i, sig_i, status)$.
Else $pk_i \leftarrow \varepsilon; sig_i \leftarrow \varepsilon; st \leftarrow \varepsilon$.
$\pi_2 \leftarrow P_2(1^\lambda, (opk, C, i, pk, cert, s),$
$(osk, r_e), R_2)$.
If $V_1(1^\lambda, (opk, gmpk, m, C), \pi_1, R_1) = 0$
then return $(0, \varepsilon, 0)$.
If $pk \neq pk_i$ or $reg[i] = \varepsilon$ or status $= 0$
then return $(0, \varepsilon, 0)$.
$\tau \leftarrow (pk_i, sig_i, i, pk, cert, s, \pi_2)$.
Return $(i, \tau,$ st$)$.

IsActive$(i, reg) \to (0/1)$

If $reg[i] \neq \varepsilon$ then
Parse $reg[i]$ as $(pk_i, sig_i, status)$.
Return $status$.

UKg$(1^\lambda) \to (\mathbf{upk}, \mathbf{usk})$

$(\mathbf{upk}, \mathbf{usk}) \xleftarrow{\$} K_s(1^\lambda)$.
Return ($\mathbf{upk}$, $\mathbf{usk}$).

Revoke$(i, \mathbf{grt}[i], \mathbf{ik}, RL, reg) \to (RL, reg)$

Parse $\mathbf{grt}[i]$ as $(usk[i],\ pk[i])$
Query $reg[i] \to (i, pk_i,\ st)$
If ($st \neq 0$ and $pk_i = pk[i]$)
then $RL \leftarrow RL \cup (usk[i], pk[i])$
update $reg[i]$ to inactive;
Return RL, reg.

Verify$(\mathbf{gpk}, m, \sigma, RL) \to 1/0$

Parse $\mathbf{gpk}$ as $(1^\lambda, R_1, R_2, opk, gmpk)$.
Parse σ as $(C, \pi_1, \mathbf{grt}[i])$.
Parse $\mathbf{grt}[i]$ as $(usk[i], pk_i)$.
If $V_1(1^\lambda, (opk, gmpk, m, C), \pi_1, R_1) = 1$
and $(usk[i], pk_i) \notin RL$ then return 1
else return 0.

Judge$(\mathbf{gpk}, i, \mathbf{upk}[i], m, \sigma, \tau) \to 1/0$

Parse $\mathbf{gpk}$ as $(1^\lambda, R_1, R_2, opk, gmpk)$.
Parse σ as $(C, \pi_1, \mathbf{grt}[i])$.
If (i, τ, st) = $(0, \varepsilon, 0)$ then
return $V_1(1^\lambda, (opk, gmpk, m, C), \pi_1, R_1) =$
0.
Parse τ as $(\bar{pk}, \bar{sig}, i', pk, cert, s, \pi_2)$.
If $V_2(1^\lambda, (C, i', pk, cert, s), \pi_2, R_2) = 0$
then return 0
If all of the following are true then return
1 else return 0:
-$i = i'$
-$\mathsf{Vf}(\mathbf{upk}[i], \bar{pk}, \bar{sig})$
-$\bar{pk} = pk$.

Fig. 2 Algorithms of the new fully dynamic group signature scheme

algorithm, which is used in schemes like [16], use of the implicit tracing algorithm increases the time consumption. Hence, instead of using the implicit tracing algorithm given in *VLR*, we use algorithms provided in [3] for our scheme's tracing mechanism.

As well, *VLR* manages a token system. Thus, our scheme should consist user tokens and those tokens should be unique to the users. Furthermore, tokens should not reveal user's identity in case of disclosing to the outsiders. We generate tokens for members, which will not expose identity of the members even though tokens are opened to the outsiders. We use the combination of each group member's personal secret key and his verification key as his token, and we maintain the list *RL* with revoked members' tokens.

3.2 Description of Our Scheme

There are two authorities, *group manager* and *opener*. The trusted setup is responsible for generating the group public key and keys for the authorities. The *group manager* manages member registration and member revocation while the *opener* traces signers.

When a new user wants to join the group, he interacts with the group manager via *group-joining protocol* (Fig. 1), which allows new users to generate their public key and secret keys. We assume this interaction between the new user, and the group manager is done through a secure channel. The new user produces a signature on his verification key and sends both the signature and the key to the group manager. If the signature is acceptable, then the group manager accepts him as a new member. In the registration table *reg*, we maintain a field called *Status* for each member to identify the active status of them. Thus, the group manager stores the index i, verification key pk_i, and the signature sig_i of the new member in *reg* and makes the status of the new member as active. After that, the group manager issues member certification to the new member. Now the new member can generate signatures on messages using his secret key.

Each member has a unique token, which is the tracing key to identify the validity of signers, whether they are revoked or not. Here we use the member's personal secret key $usk[i]$ and his verification key pk_i as the token since $usk[i]$ or pk_i does not help to reveal any other information. We check the existence of the new user keys against *reg* at the joining protocol. Thus, in a situation that a revoked member wants to join again, he cannot use his previous keys, and he has to follow the process as a new user. That is to secure the scheme against adversaries who steal tokens and try to join the group. During the member revocation, the group manager adds the revoking member's token to *RL* and updates *reg* to inactive. When a member needs to sign a message, he generates the signature on a message with his secret key and passes to the verifier with his token for verification. The verifier authenticates the signature on the given message and checks the validity of the signer with the provided token against the latest *RL*. In the case of necessity to trace the signer, the opener can trace the signer using opener's key, and he can check the status of the signer in *reg*.

Our scheme is a tuple $FDGS = (\mathsf{GKg}, \mathsf{UKg}, \mathsf{Join}, \mathsf{Issue}, \mathsf{Revoke}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Open}, \mathsf{Judge})$, which consists of polynomial-time algorithms. Each algorithm is described in below. GKg, UKg, and Judge are same as in [3] and Join, Issue, Sign, Verify, and Open are different from the algorithms given in [3] since we have to generate and pass the member's token as an additional attribute in our scheme. Revoke helps to revoke the misbehaved users.

- $\mathsf{GKg}(1^\lambda)$: On input 1^λ, the trusted party obtains a group public key **gpk** and authority keys, **ik** and **ok**. Then gives secret keys, **ik** to the group manager and **ok** to the opener.
- $\mathsf{UKg}(1^\lambda)$: Every user who wants to be a member should run this algorithm before the *group-joining protocol* to obtain their personal public key and personal private key (**upk**$[i]$, **usk**$[i]$). UKg takes as input 1^λ. We assume **upk** is publicly available.
- Join, Issue: The *group-joining protocol* is an *interactive protocol* between the group manager and the user who wants to be a member. Join is implemented by the user while Issue is implemented by the group manager. Join allows new users to generate keys and a signature on the keys which are needed to join the group. Issue allows the group manager to validate the keys and the signatures sent by users and generate member certifications. Each algorithm takes an incoming message as input and returns an outgoing message. Join and Issue maintains their current status for both parties. The user i generates a public / secret key pair pk_i and sk_i. Then, he produces a signature sig_i on pk_i using **usk**$[i]$, which was obtained in UKg. Then, user sends sig_i and pk_i to the group manager to authenticate. The group manager authenticates the signature sig_i on pk_i and generates member certification by signing pk_i with his private key **ik** (**gmsk**). The group manager stores new member's informations, i, pk_i, and sig_i with the *status* as 1 (active) in *reg*. Then, he sends member certification $cert_i$ to the user who is the new member of the group. After that new user can make his secret key **gsk**$[i] = (i, pk_i, sk_i, cert_i)$, and his token **grt**$[i] =$ (**usk**$[i]$, pk_i).
- Revoke(i, **grt**$[i]$, **ik**, *RL*, *reg*): This algorithm takes, index i of the member, who wants to be revoked and the group manager's secret key **ik** as inputs. First, the group manager queries *reg* using the index i to obtain the information of the member stored. Then, he checks whether the queries are equal to the data obtained by parsing the **grt**$[i]$. If the data are equal and if the user is active, insert ($usk[i]$, pk_i) to *RL* and updates *reg* to 0 (inactive).
- Sign(**gpk**, **gsk**$[i]$, **grt**$[i]$, m): This randomized algorithm generates a signature σ on a given message m. It takes the group public key **gpk**, the group member's secret key **gsk**, and the message m as inputs. In addition, we pass the group member's token as an input to prove that the member is an active person at the time of signing.
- Verify(**gpk**, m, σ, RL): This deterministic algorithm allows anyone in possession of group public key **gpk** to verify the given signature σ on the message m and checks the validity of the signer against *RL*. This algorithm outputs 1 if both conditions are valid. Otherwise, it returns 0.
- Open(**gpk**, **ok**, *reg*, m, σ): This deterministic algorithm traces the signer by taking **gpk**, the opener's secret key **ok**, *reg*, the message m, and the signature σ as inputs.

It returns the index of the signer, the proof of the claim τ, and the status of the signer *st* at *reg*. If the algorithm failed to trace the signature to a particular group member, it returns $(0, \varepsilon, 0)$.

– Judge(**gpk**, i, **upk**[i], m, σ, τ): This deterministic algorithm outputs either 1 or 0 depending on the validity of the proof τ on σ. This takes, the group public key **gpk**, the member index i, the tracing proof τ, the member verification key **upk**[i], the message m, and the signature σ as inputs. The algorithm outputs 1 if τ can proof that i produced σ. Otherwise, it returns 0.

In addition, we use the following simple polynomial-time algorithm.

– IsActive(i,*reg*): This algorithm determines whether the member i is active by querying the registration table and outputs either 0 or 1.

4 Security Notions of the Scheme

Even though the BMW03 model has two key requirements, full anonymity and full traceability, the scheme in [3] has three key requirements; anonymity, traceability, and non-frameability. Since *full traceability* discussed in the BMW03 model covers both traceability and non-frameability, the BMW03 model has only two requirements. In the setting of [3], traceability and non-frameability are separated since non-frameability can be achieved with lower levels of trust in the authorities than traceability as discussed below. According to the scheme in [3], the opener's secret key is provided to an adversary in traceability game but, the issuer's secret key is not provided. The scheme in [3], they assume that the opener is partially corrupted in traceability. But in non-frameability, both the opener's and the tracer's secret keys are given to the adversary. Thus, the adversary is stronger in non-frameability than in traceability. Thus, non-frameability is separated from the traceability in [3]. Moreover, anonymity allows the adversary to corrupt the issuer in [3]. Thus, we provide the issuer's secret key to the adversary but not the opener's secret key.

However, the scheme in [3] does not support member revocation but our scheme supports. Thus, we adapt the security experiments and the oracles to be compatible with *VLR*. Before we discuss the security notions, we define the set of oracles that we use. We suggest a new oracle, revoke to maintain the member revocation queried by any adversary.

For the requirement of anonymity, we suggest a restricted version of full anonymity. In the full-anonymity game, we provide all the members' secret keys to the adversary including challenged indexes' keys to the adversary. In our scheme, this may help the adversary to create the challenged indexes' tokens since he knows all the members' personal secret keys (*usk*) and he can execute Verify to check which index is used to generate the challenged signature. Thus, we will not provide users' personal secret keys to the adversary when he requests for user's secret keys. However, he can request for any private signing key. Hence, we suggest a new security notion *almost-full anonymity* to show the security of our scheme. Since the almost-full anonymity

does not allow members' personal secret keys to the adversary, it is somewhat weaker than the full anonymity, and since, it provides members' secret signing keys including challenged indices' to the adversary, it is stronger than the selfless-anonymity.

4.1 The Oracles

All the oracles that we use are specified in Fig. 3. We maintain a set of global lists, which are manipulated by the oracles in the security experiments discussed later. **HUL** is the honest user list, which maintains the indexes of the users who are added to the group. When the adversary corrupts any user, that user's index is added to **CUL**. **SL** carries the signatures that obtained from Sign oracle. When the adversary requests a signature, the generated signature, the index, and the message are added to **SL**. When the adversary accesses Challenge oracle, the generated signature is added to **CL** with the message sent. We use a set S to maintain a set of revoked users.

- AddU(i): The adversary can add a user $i \in \mathbb{N}$ to the group as an honest user. The oracle adds i to **HUL** and selects keys for i. It then executes the *group-joining protocol*. If Issue accepts, then adds the state to *reg* and if Join accepts then generates **gsk**[i]. Finally, returns **upk**[i].
- CrptU(i, upk): The adversary can corrupt user i by setting its personal public key **upk**[i] to upk. The oracle adds i to **CUL** and initializes the issuer's state in *group-joining protocol*.
- SendToIssuer(i, M_{in}): The adversary acts as i and engages in *group-joining protocol* with Issue-executing issuer. The adversary provides i and M_{in} to the oracle. The oracle which maintains the Issue state returns the outgoing message and adds a record to *reg*.
- SendToUser(i, M_{in}): The adversary corrupts the issuer and engages in *group-joining protocol* with Join-executing user. The adversary provides i and M_{in} to the oracle. The oracle which maintains the user i state, returns the outgoing message, and sets the private signing key of i to the final state of Join.
- RevealU(i): The adversary can reveal secret keys of the user i. We only provide user's private signing key **gsk**[i] not his personal private key **usk**[i].
- ReadReg(i): The adversary can read the entry of i in *reg*.
- ModifyReg(i, val): The adversary can modify the contents of the record for i in *reg* by setting val.
- Sign(i, m): The adversary obtains a signature σ for a given message m and user i who is an honest user and has private signing key.
- Chal$_b$(i_0, i_1, m): This oracle is for defining anonymity and provides a group signature for the given message m under the private signing key of i_b, as long as both i_0, i_1 are active and honest users having private signing keys.
- Revoke(i): The adversary can revoke user i. The oracle updates the record for i in *reg* and adds revocation token of i to the set S.

AddU(i)
If $i \in \mathbf{HUL} \cup \mathbf{CUL}$, then return ε.
$\mathbf{HUL} \leftarrow \mathbf{HUL} \cup \{i\}$
$\mathbf{gsk}[i] \leftarrow \varepsilon$; $\mathbf{grt}[i] \leftarrow \varepsilon$
$dec^i_{is} \leftarrow$ cont;
$(\mathbf{upk}[i], \mathbf{usk}[i]) \leftarrow \mathsf{UKg}(1^\lambda)$
$St^i_{jn} \leftarrow (\mathbf{gpk}, \mathbf{upk}[i], \mathbf{usk}[i])$
$St^i_{is} \leftarrow (\mathbf{gpk}, \mathbf{ik}, i, \mathbf{upk}[i])$
$M_{jn} \leftarrow \varepsilon$.
$(St^i_{jn}, M_{is}, dec^i_{is}) \leftarrow \mathsf{Join}(St^i_{jn}, M_{jn})$.
While (dec^i_{is} = cont and dec^i_{jn} = cont) then do
$(St^i_{is}, M_{jn}, dec^i_{is}) \leftarrow \mathsf{Issue}(St^i_{is}, M_{is})$.
$(St^i_{jn}, M_{is}, dec^i_{jn}) \leftarrow \mathsf{Join}(St^i_{jn}, M_{jn})$.
End while.
If $dec^i_{is} =$ accept then $reg[i] \leftarrow (St^i_{is}, 1)$
If $dec^i_{jn} =$ accept then $\mathbf{gsk}[i] \leftarrow (St^i_{jn})$ and $\mathbf{grt}[i] = \mathbf{usk}[i]$
Return $\mathbf{upk}[i]$

CrptU(i, upk)
If $i \in \mathbf{HUL} \cup \mathbf{CUL}$ then return ε.
$\mathbf{CUL} \leftarrow \mathbf{CUL} \cup \{i\}$;
$\mathbf{upk}[i] \leftarrow upk$.
$dec^i_{is} \leftarrow$ cont.
$St^i_{is}(\mathbf{gpk}, \mathbf{ik}, i, \mathbf{upk}[i])$
Return 1.

ReadReg(i)
Return $(reg[i])$

ModifyReg(i, val)
$reg[i] \leftarrow val$

Revoke(i)
If $i \notin \mathbf{HUL}$ then return ε.
If $i \in \mathbf{CL}$ then return ε.
If $\mathsf{IsActive}(i, \text{reg}) = 0$ then return ε.
$S = S \cup \{\mathbf{grt}[i]\}$
update $reg[i]$
Return 1

SendToIssuer(i, M_{in})
If $i \notin \mathbf{CUL}$ then return ε
If $dec^i_{is} \neq cont$ then return ε
If $i \notin \mathbf{HUL}$ then return ε.
$St^i_{is} \leftarrow (\mathbf{gpk}, ik, i, \mathbf{upk}[i])$
$(St^i_{is}, M_{out}, dec^i_{is}) \leftarrow \mathsf{Issue}(St^i_{is}, M_{in})$.
If $dec^i_{is} =$ accept then
$reg[i] \leftarrow (St^i_{is}, 1)$
Return (M_{out}, dec^i_{is})

Sign(i, m)
If $i \notin \mathbf{HUL}$ then return ε.
If $\mathbf{gsk}[i] = \varepsilon$ then return ε.
If $\mathsf{IsActive}(i, \text{reg}) = 0$ then return ε
$\sigma = \mathsf{Sign}(\mathbf{gpk}, \mathbf{gsk}[i], \mathbf{grt}[i], m)$
$\mathbf{SL} = \mathbf{SL} \cup \{(i, m, \sigma)\}$
Return σ

SendToUser(i, M_{in})
If $i \in \mathbf{CUL}$ then return ε.
If $i \notin \mathbf{HUL}$ then
$\mathbf{HUL} \leftarrow \mathbf{HUL} \cup \{i\}$
$\mathbf{gsk}[i] \leftarrow \varepsilon$; $M_{in} \leftarrow \varepsilon$
$(\mathbf{upk}[i], \mathbf{usk}[i]) \leftarrow \mathsf{UKg}(1^\lambda)$
If $dec^i_{jn} \neq$ cont then return ε
$St^i_{jn} \leftarrow (\mathbf{gpk}, \mathbf{upk}[i], \mathbf{usk}[i])$
$(St^i_{jn}, M_{out}, dec^i_{jn}) \leftarrow \mathsf{Join}(St^i_{jn}, M_{in})$
If $dec^i_{jn} =$ accept then $\mathbf{gsk}[i] \leftarrow St^i_{jn}$ and $\mathbf{grt}[i] = \mathbf{usk}[i]$
Return (M_{out}, dec^i_{jn})

Chal$_b$(i_0, i_1, m)
If $i_0 \notin \mathbf{HUL}$ or $i_1 \notin \mathbf{HUL}$ then return ε.
If $\mathbf{gsk}[i_0] = \varepsilon$ or $\mathbf{gsk}[i_1] = \varepsilon$ then return ε.
If $\mathsf{IsActive}(i_0, reg) = 0$ or $\mathsf{IsActive}(i_1, reg) = 0$ then return ε
$\sigma = \mathsf{Sign}(\mathbf{gpk}, \mathbf{gsk}[i_b], \mathbf{grt}[i_b], m)$
$\mathbf{CL} = \mathbf{CL} \cup \{(m, \sigma)\}$
Return σ

RevealU(i)
If $i \notin \mathbf{HUL} \setminus (\mathbf{CUL} \cup \mathbf{CL})$ then return ε.
Return $\mathbf{gsk}[i]$

Open(m, σ)
If $(m, \sigma) \in \mathbf{CL}$ then return $(\varepsilon, \varepsilon, \varepsilon)$
If $\mathsf{Verify}(\mathbf{gpk}, m, \sigma, S) = 0$ then return $(\varepsilon, \varepsilon, \varepsilon)$
Return $\mathsf{Open}(\mathbf{gpk}, \mathbf{osk}, reg, m, \sigma)$

Fig. 3 Oracles

– $\mathsf{Open}(m, \sigma)$: The adversary can access this *opening* oracle with a message m and a signature σ to obtain the identity of the user, who generated the signature σ. If σ is queried before for Chal_b, oracle will abort.

4.2 Correctness

The notion of correctness requires that any signature generated by any honest and active users should be valid and Open should correctly identify the signer for a given message and a signature. Moreover, the proof returned by Open should be accepted by Judge. Hence, any scheme is correct if the advantage of the correctness game is 0, for all $\lambda \in \mathbb{N}$ and for any adversary A.

We let, $\mathbf{Adv}^{corr}_{FDGS,A}(\lambda) = \Pr[\mathbf{Exp}^{corr}_{FDGS,A}(\lambda) = 1]$.

$\underline{\mathbf{Exp}^{corr}_{FDGS,A}(\lambda)}$

$(\mathbf{gpk}, \mathbf{ok}, \mathbf{ik}) \leftarrow \mathsf{GKg}(1^\lambda)$; $\quad \mathbf{HUL} \leftarrow \emptyset$; $\quad (i, m) \leftarrow A(\mathbf{gpk}; \mathsf{AddU}, \mathsf{ReadReg}, \mathsf{Revoke})$;
If $i \notin \mathbf{HUL}$ or $\mathbf{gsk}[i] = \varepsilon$ or $\mathsf{IsActive}(i, reg) = 0$, then return 0.
$\sigma \leftarrow \mathsf{Sign}(\mathbf{gpk}, \mathbf{gsk}[i], m)$;
If $\mathsf{Verify}(\mathbf{gpk}, m, \sigma, S) = 0$, then return 1.
$(i', \tau) \leftarrow \mathsf{Open}(\mathbf{gpk}, \mathbf{ok}, reg, m, \sigma)$;
If $i \neq i'$, then return 1.
If $\mathsf{Judge}(\mathbf{gpk}, i, \mathbf{upk}[i], m, \sigma, \tau) = 0$, then return 1 else return 0.

4.3 Anonymity

The anonymity requires the signatures do not reveal the identity of the signer. In the anonymity game, the adversary's goal is to identify the index that is used to create the signature. We allow the adversary A to corrupt any user and allow him to fully corrupt the group manager. Also, A can learn secret signing keys of any user. In full-anonymity game, adversary can access all the secret keys of any member. However, we suggest a new security notion *almost-full anonymity*, which does not allow to reveal the personal secret keys of the users since the adversary can create the tokens of the challenged ones and check with Verify. Hence, he can easily win the game. We say that *FDGS* scheme is almost-fully anonymous if the advantage of the adversary $\mathbf{Adv}^{anon}_{FDGS,A}(\lambda)$ is negligible for any polynomial-time adversary.

In the game, A selects two active group members and a message to challenge the game. He has to guess which member is used to generate the signature. He wins if he can guess the member correctly. We allow only one guess.

We let $\mathbf{Adv}^{anon}_{FDGS,A}(\lambda) = \Pr[\mathbf{Exp}^{anon\text{-}0}_{FDGS,A}(\lambda) = 1] - \Pr[\mathbf{Exp}^{anon\text{-}1}_{FDGS,A}(\lambda) = 1]$.

$\mathbf{Exp}^{anon\text{-}b}_{FDGS,A}(\lambda)$
$(\mathbf{gpk}, \mathbf{ok}, \mathbf{ik}) \leftarrow \mathsf{GKg}(1^\lambda)$; $\mathbf{HUL}, \mathbf{CUL}, \mathbf{SL}, \mathbf{CL} \leftarrow \emptyset$;
$b^* \leftarrow A(\mathbf{gpk}, \mathbf{ik}; \mathsf{CrptU}, \mathsf{SendToUser}, \mathsf{RevealU}, \mathsf{Open}, \mathsf{ModifyReg}, \mathsf{Revoke}, \mathsf{Chal}_b)$;
Return b^*;

4.4 Non-Frameability

The non-frameability ensures that any adversary unable to produce a signature can be attributed to an honest member, who did not produce it.

We let $\mathbf{Adv}^{non\text{-}fram}_{FDGS,A}(\lambda) = \Pr[\mathbf{Exp}^{non\text{-}fram}_{FDGS,A}(\lambda) = 1]$.

In this game, we only require that the framed member is honest. Thus, the adversary A can fully corrupt the group manager and the opener.

Formally, the *FDGS* scheme is non-frameable for all $\lambda \in \mathbb{N}$ and for any adversary A.

$\mathbf{Exp}^{non\text{-}fram}_{FDGS,A}(\lambda)$
$(\mathbf{gpk}, \mathbf{ok}, \mathbf{ik}) \leftarrow \mathsf{GKg}(1^\lambda)$; $\mathbf{HUL}, \mathbf{CUL}, \mathbf{SL} \leftarrow \emptyset$;
$(m, \sigma, i, \tau) \leftarrow A(\mathbf{gpk}, \mathbf{ik}, \mathbf{ok}; \mathsf{CrptU}, \mathsf{SendToUser}, \mathsf{RevealU}, \mathsf{Sign}, \mathsf{ModifyReg})$;
If $\mathsf{Verify}(\mathbf{gpk}, m, \sigma, S) = 0$, then return 0.
If $\mathsf{Judge}(\mathbf{gpk}, i, \mathbf{upk}[i], m, \sigma, \tau) = 0$, then return 0.
If $i \notin \mathbf{HUL}$ or $(i, m, \sigma, \tau) \in \mathbf{SL}$, then return 0 else 1.

4.5 Traceability

The traceability requires any adversary cannot produce a signature that unable to identify the origin of the signature. That means the adversary's challenge is to generate a signature that cannot be traced to an active member of the group. In this game, A is allowed to corrupt any user and he has the opener's key, but he is not allowed to corrupt the group manager since he can produce dummy users. He wins if he can create a signature, whose signer cannot be identified or signer is an inactive member when creating the signature, or Judge algorithm does not accept the Open algorithm's decision.

We let $\mathbf{Adv}^{trace}_{FDGS,A}(\lambda) = \Pr[\mathbf{Exp}^{trace}_{FDGS,A}(\lambda) = 1]$.

$\mathbf{Exp}^{trace}_{FDGS,A}(\lambda)$
$(\mathbf{gpk}, \mathbf{ok}, \mathbf{ik}) \leftarrow \mathsf{GKg}(1^\lambda)$; $\mathbf{HUL}, \mathbf{CUL}, \mathbf{SL} \leftarrow \emptyset$;
$(m, \sigma) \leftarrow A(\mathbf{gpk}, \mathbf{ok}; \mathsf{AddU}, \mathsf{CrptU}, \mathsf{SendToIssuer}, \mathsf{RevealU}, \mathsf{Sign}, \mathsf{ModifyReg}, \mathsf{Revoke})$;
If $\mathsf{Verify}(\mathbf{gpk}, m, \sigma, S) = 0$, then return 0.
$(i, \tau) \leftarrow \mathsf{Open}(\mathbf{gpk}, \mathbf{ok}, reg, m, \sigma)$;
If $i = 0$ or $\mathsf{Judge}(gpk, i, upk[i], m, \sigma, \tau) = 0$, then return 1 else return 0.

5 Security Proof of Our Scheme

We can prove that our scheme is anonymous, non-frameable and traceable according to the experiments described above and which are discussed in [3] and [7]. Even though our scheme has used a token system as an additional attribute than the scheme in [3], since we are not providing the tokens to the adversary and since we have used the member's personal secret key **usk**[i] and his verification key $\mathbf{pk}_i$ as his revocation token, which cannot be used to learn about the member, there is no impact on the security of the scheme from the token system. Since our scheme requires a reasonable and sufficient security notion for the problem of considering full anonymity, we use almost-full anonymity and we use security experiments provided above instead of experiments given in [3]. However, due to the page limitation, we provide only a summary of security proof and we will give a detailed proof of security in a full version of this paper.

5.1 Anonymity

On the assumption that P_1 is computational zero knowledge for ρ_1 over Dom_1 and P_2 is computational zero knowledge for ρ_2 over Dom_2, two simulations Sim_1 and Sim_2 can be fixed as $\Pi_1 = P_1, V_1, Sim_1$; $\Pi_2 = P_2, V_2, Sim_2$; Π_1 and Π_2 are the simulation-sound zero-knowledge non-interactive proof systems of them for $L_{\rho 1}$ and $L_{\rho 2}$, respectively.

For any polynomial-time adversary B, who will challenge the anonymity of our scheme and who can construct polynomial-time IND-CCA adversaries A_0, A_1 against encryption scheme E, an adversary A_s against the simulation soundness of Π and distinguishers D_1, D_2 that distinguish real proofs of Π_1 and Π_2, respectively, for all $\lambda \in \mathbb{N}$, we say

$$\mathbf{Adv}^{anon}_{FDGS,B}(k) \leq \mathbf{Adv}^{ind-cca}_{E,A_0}(k) + \mathbf{Adv}^{ind-cca}_{E,A_1}(k) + \mathbf{Adv}^{ss}_{\Pi,A_s}(k) + 2 \cdot (\mathbf{Adv}^{zk}_{P_1,Sim_1,D_1}(k) + \mathbf{Adv}^{zk}_{P_2,Sim_2,D_2}(k)).$$

According to the Lemma 5.1 described and proved in [3], we can say the left side function is negligible since all the functions on the right side are negligible under the assumptions on the security of building blocks described. This proves the anonymity of our scheme.

5.2 Non-Frameability

If there is a non-frameability adversary B, who creates at most $n(k)$ honest users, where n is a polynomial and who constructs two adversaries A_2, A_3 against the digital

signature scheme, on the assumption that (P_1, V_1), (P_2, V_2) are sound proof systems for ρ_1, ρ_2, respectively, we say

$$\mathbf{Adv}^{non-fram}_{FDGS,B}(k) \leq 2^{-k+1} + n(k) \cdot (\mathbf{Adv}^{unforg-cma}_{DS,A_2}(k) + \mathbf{Adv}^{unforg-cma}_{DS,A_3}(k)).$$

On the assumption that the scheme *DS* is secure, all the functions on the right side are negligible, so the left side function. Thus, our scheme is non-frameable according to the definition of *DS*.

5.3 Traceability

If there is a traceability adversary B, who constructs an adversary A_1 against the scheme *DS*, on the assumption that (P_1, V_1) is a sound proof system for ρ_1, we say

$$\mathbf{Adv}^{trace}_{FDGS,B}(k) \leq 2^{-k+1} + \mathbf{Adv}^{unforg-cma}_{DS,A_1}(k).$$

On the assumption that *DS* is secure against traceability, all the functions on the right side are negligible. Because of this, the advantage of B is negligible. Thus, it proves that our scheme is traceable.

6 Conclusion

In this paper, we have presented a simple fully dynamic group signature scheme that can be used as a basic scheme to develop with different approaches. We have constructed our scheme based on the scheme in [3] and proposed *Verifier-Local revocation* mechanism, which ease member revocation and convenient for large groups. Thus, our scheme is more flexible and suitable for dynamically changing groups, even they are large. We have shown how to achieve the security with almost-fully anonymity, which is a limited version of fully anonymity.

Acknowledgements This work is supported in part by JSPS Grant-in-Aids for Scientic Research (A) JP16H01705 and for Scientic Research (B) JP17H01695.

References

1. Bellare, M., Micali, S.: How to sign given any trapdoor function. In: CRYPTO 1988, vol. 403, pp. 200–215. LNCS (1988)
2. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: formal definitions, simplified requirements, and a construction based on general assumptions. In: EUROCRYPT 2003, vol. 2656, pp. 614–629. LNCS (2003)

3. Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: the case of dynamic groups. In: CT-RSA 2005, vol. 3376, pp. 136–153. LNCS (2005)
4. Blum, M., De Santis, A., Micali, S., Persiano, G.: Noninteractive zero-knowledge. SIAM J. Comput. **20**(6), 1084–1118 (1991)
5. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: CRYPTO 2004, vol. 3152, pp. 41–55. LNCS (2004)
6. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: ACM-CCS 2004, pp. 168–177. ACM (2004)
7. Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J.: Foundations of fully dynamic group signatures. In: ACNS 2016, pp. 117–136. LNCS (2016)
8. Bresson, E., Stern, J.: Efficient revocation in group signatures. In: PKC 2001, vol. 1992, pp. 190–206. LNCS (2001)
9. Brickell, E.: An efficient protocol for anonymously providing assurance of the container of the private key. Submitted to the Trusted Computing Group (April 2003)
10. Camenisch, J., Groth, J.: Group signatures: better efficiency and new theoretical aspects. In: SCN 2004, vol. 3352, pp. 120–133. LNCS (2004)
11. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: CRYPTO 2002, vol. 2442, pp. 61–76. LNCS (2002)
12. Chaum, D., van Heyst, E.: Group signatures. In: EUROCRYPT 1991, vol. 547, pp. 257–265. LNCS (1991)
13. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM Rev. **45**(4), 727–784 (2003)
14. Kiayias, A., Yung, M.: Secure scalable group signature with dynamic joins and separable authorities. Int. J. Secur. Netw. **1**(1–2), 24–45 (2006)
15. Libert, B., Ling, S., Mouhartem, F., Nguyen, K., Wang, H.: Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In: ASIACRYPT 2016, vol. 10032, pp. 373–403. LNCS (2016)
16. Ling, S., Nguyen, K., Wang, H.: Group signatures from lattices: simpler, tighter, shorter, ring-based. In: PKC 2015, vol. 9020, pp. 427–449. LNCS (2015)
17. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: FOCS 1999. pp. 543–553. IEEE (1999)
18. Song, D.X.: Practical forward secure group signature schemes. In: ACM-CCS 2004, pp. 225–234. ACM (2001)