Democratic Group Signatures on Example of Joint Ventures

This is a full version of the fast abstract which appears at ASIACCS'2006.

Mark Manulis

Horst-Görtz Institute
Ruhr-University of Bochum
D-44801, Germany
EMail: mark.manulis@rub.de

Abstract. In the presence of economic globalization joint venture is one of the most common and effective means of conducting business internationally. By building joint ventures companies form strategic alliances that help them to enter new economic markets and further their business goals in a cooperative effort without loosing own independence. Upon building a joint venture company, two or more ”parent” companies agree to share capital, technology, human ressources, risks and rewards in a formation of a new entity under shared control by a ”board of directors”, which consists of representatives of ”parent” companies. The establishment of such shared control is tricky and relies generally on the ”trust, but verify” relationship, i.e., companies trust the information they receive from prospective partners, but it is a good business practice to verify the facts. In this paper we focus on the issue of the shared financial control in a joint venture. We consider the mostly preferred form of the control where every member of the board is able to issue payment orders on behalf of the joint venture, but at the same time representatives of other companies, should be able to monitor the accounting to achieve fairness in the spending of shared funds. For this form of the shared control we propose a new secure group-oriented signature scheme, called a *democratic group signature* scheme, which results from the modification of the standard notion of group signatures by eliminating the role of the group manager. We also show that existing schemes, e.g., ring and group signatures, cannot be used to realize the required shared control based on the ”trust, but verify” relationship.
Key words: group-oriented signatures, ”trust, but verify”, anonymity, joint membership control, individual tracing

1 Introduction

Joint ventures (JV) become an increasingly common way for companies to form strategic alliances for the means of economic expansion and realization of new business plans, such as entering new economic markets, developement of new technology or conducting trade internationally. A joint venture arrangement between partner companies has one of the following two forms: a joint venture agreement (JVA) or a joint venture company (JVCo). JVA is a simple agreement between participating companies which determines what each will bring to and gain from the project without forming a separate legal entity, and is considered to be a ”once more” arrangement that is established for the mutual benefit of partners, e.g., arrangement on a joint distribution network or shared product facilities. JVCo goes a step further in setting up a company in order to procure the project. It is usually specified by own trade and business plans, has some independence to pursue own commercial strategy with agreed objectives, and is able to access the market with own trade mark.

Depending on the arrangement the JVCo may be open for other companies to join if they fulfill certain criteria and agree to sign the contract. Usually, current JV partners decide together whether a new company is allowed to join the JVCo. Similarly, a company may resign from the JV contract and leave the JVCo. Criteria for join and leave are usually defined in a common policy which is agreed by ”parent” companies and is part of the signed contract.

Where a JV company is set up the control of the company is generally proportional to the percentage of shares held by each ”parent” company, which in turn reflects the investment by the companies, respectively. The so-called ”board of directors” of JVCo is legally responsible for the supervision/strategic plans and active management of JVCo on the basis of a shared control, and consists usually of representatives of ”parent” companies. The organization of the shared control in JVCo is a challenging task, since a specific trust relationship between ”parent” companies, called ”trust, but verify” has to be considered. The ”trust, but verify” relationship is followed from the natural objectives of ”parent” companies to gain as much as possible from entering a joint venture while keeping the occuring expenses as little as possible. Therefore, by the means of cooperation the JV partners generally trust the information they receive from each other, but should be able to verify the facts.

Our paper focuses on the establishment of the shared financial control in a JVCo, however, we remark that our model may also be utilized for other shared control issues. We consider a simpler case of a JV arrangement where all ”parent” companies provide equal contribution to the shared budget, and have, therefore, an equal number of representatives in the board, e.g., one. The financial control in the JVCo includes among other topics the issuing of payment orders on behalf of JVCo and the monitoring of the accounting of the company. We are not interested in the organization of the JVCo, e.g., whether there is a separate finance department, and assume for simplicity that the responsibility for the financial control is given to the directors, i.e., members of the board. The organization of the financial control in JVCo is a subject of the JV contract which is agreed and signed by all ”partner” companies. Since all JV companies have individual arrangements there is no common form which is applied in all JVCos. However, the following form of the shared financial control seems to be mostly fair for all JV participants, and is, therefore, preferred in the most arrangements: ”parent” companies agree that each of them independently is allowed to issue payment orders from the shared budget of the JVCo for the amount which does not exceed their own contribution to this budget. Obviously, this form alone provides less flexibility, since sometimes companies need to make investments which are higher than their contributions. For this case an additional clause is usually taken into the JV arrangement: whenether a company issues a payment order for the amount which exceeds own contribution, it is obliged to refund the difference to the JVCo’s budget.

In this paper we propose a security model for the described form of the financial control in JVCos. According to the ”trust, but verify” relationship ”parent” companies should be able to monitor the accounting of the JVCo, which in turn means that representatives in the ”board of directors” should be given a power for the independent verification of issued payment orders signed by other directors and ability to create own view on the actual state of all JV partners’ debts to the shared JVCo’s budget.

Another security aspect of this form of the shared financial control is the anonymity of directors who issue the payment orders. For a JVCo the own establishment on the market is of great importance, especially, when JVCo enters the market with its own trademark. Hence, it is important to hide the information which concerns the affiliation of the representative who has issued the payment order to his ”parent” company from parties who get this money and are not involved in the JV arrangement. In other words, if a third party receives a payment order signed on behalf of the JVCo then it must not be able to link the signature to the ”parent” company whose representative has signed the order. Note that for the means of the shared monitoring of accounting other JV partners should be able to identify the ”parent” company (i.e., its representative) from the signature of the payment order.

Our model consists of a new group-oriented signature scheme, which results from the modification of the standard notion of group signatures. The main changes to the classical model of group signatures are: the absence of a central trusted authority (usually called a group manager), and the ability of the individual tracing of signatures, i.e., every member of the group (director) is able to open signatures created by other members. In Section 1.1 we describe in detail why the standard model of group signatures, and another related group-oriented signature schemes called ring signatures are not applicable to the focused scenario of the shared financial control in JVCos.

1.1 Related Work

The concept of *group signatures* was first introduced by Chaum and van Heyst [13], and further studied and improved in [11], [10], [3], [4], [15], [7], [6], [8], [9]. Classical group signatures allow group members to sign messages anonymously on behalf of the group. The anonymity is provided not only against nonmembers, but also against other group members. However, there exists a designated authority, called *group manager* that initializes the scheme, adds new group members, and is able to open group signatures, i.e., reveal the signer’s identity from the signature. Some schemes, like [10] distinguish between two group managers with respect to their responsibilities, i.e., membership manager that sets up the scheme and controls admission to the group, and revocation manager that opens (traces) the signatures. Group signature schemes can be used by employees of a company to sign documents on behalf of the company, or in electronic voting and bidding scenarios. Bellare *el. al.* have described in [4] and [6] formal models and security requirements for classical static and dynamic group signature schemes.

In the following we argue that this standard notion of group signatures is not applicable to the scenario of the shared financial control in a JVCo described in the introduction. The most disturbing factor is the role of the group manager that must be trusted by all group members. Assume that a classical group signature scheme is applied for the financial control in a JVCo. Then, there is a sole member (group manager) of the ”board of directors” who according to the group signature scheme is given a power to decide about the membership of JV partners and is able to verify all payment orders issued by other directors on behalf of the JVCo, i.e., open corresponding signatures. Obviously, it is be difficult to agree whose representative should take this role, because all ”parent” companies have equal rights. Even if such group manager is chosen, the rest of the board must trust him not to compromise the scheme, e.g., not to add other members to the group. Additionally, this kind of the centralized control contradicts to the idea of the shared financial control, because other directors are not be able to independently monitor the accounting of the shared budget. Surely, the group manager can be asked to open every signature and send a notification containing the signer’s identity to every other member. However, this does not only contradict to the ”trust, but verify” relationship, because other members have to trust that the information provided by the group manager is correct, but is also inefficient, because the group manager is the only to perform computations needed for the revocation (tracing), and it also becomes a ”single point of failure”. Additional drawbacks of such centralized management get clear in the context of the dynamic changes. Since ”parent” companies may resign from a JVCo there is a problem in case where the group manager represents a leaving JV partner. In this case another group manager must be chosen and the scheme has to be reinitialized, because classical group signature schemes assume that the group manager stays continuously in the group, and do not provide mechanisms to handle the opposite case. For these reasons classical group signatures cannot be effectively applied in the described scenario, and do not satisfy the fairness condition stated by the ”trust, but verify” relationship.

Ring signatures are another kind of group-oriented signature schemes which we consider in the context of the shared financial control in JV companies. The concept of ring signatures introduced by Rivest *et. al.* in [20] and developed further in [1], [21] and [18], has some significant differencies to the classical group signature schemes. In ring signature schemes there is no group initialization and no group manager. Thus, members do not have to perform any interactive protocols (i.e., to cooperate) to initialize the group in difference to classical group signature schemes, where group members usually perform an interactive protocol with the group manager to obtain their membership certificates that are then used in the generation of the group signatures. Thus, in the ring signature schemes any participant may specify a set of possible signers (usually by their public keys) and produce ring signatures that convince any verifier that the author belongs to this set without revealing any information that may be used to identify the author. The most important difference besides the absence of cooperation and the absence of the group manager is the requirement of unconditional anonymity, i.e., there is no revocation (tracing) authority which is able to reveal the signer’s identity from the ring signature.

This requirement of unconditional anonymity makes the use of ring signatures in the described scenario of the shared control impossible. It disallows the independent monitoring of the accounting and does not guarrantee that the shared capital is spent fairly by individual ”parent” companies. The fact that ring signatures cannot be opened allows directors to sign payment orders which cannot be linked to their ”parent” companies. Hence, directors must trust each other not to cheat while spending the shared capital, and truly refund the differences to the JVCo’s budget. Obviously, this contradicts to the required ”trust, but verify” relationship, because no independent verification is possible. Additionally, the absence of the group initialization and the fact that every member can define own set of possible signers makes the establishment of a JVCo as an independent company more difficult.

There exist currently no group-oriented signature schemes which can be effectively used for the means of the shared control based on the ”trust, but verify” relationship. Therefore, in our model we design a new scheme, which includes some properties of the existing schemes and states new requirements, such as individual tracing of signatures, and cooperation of participants for the initialization of the scheme, and its maintenance upon possible dynamic changes.

2 Democratic Group Signatures

In this section we present our model of a group-oriented signature scheme which can be used for the means of the described scenario of the shared control based on the ”trust, but verify” relationship. To emphasize that the ”parent” companies have equal rights in the formation and extension of a JVCo, in the issuing of payment orders, and in the monitoring of the accounting we call our scheme a *democratic group signature* scheme (DGS).

2.1 Preliminaries

Roles and Definitions. In order to link the description of our model to the example of the shared financial control in the introduction we use the following notations: *group* stands for the ”board of directors” of a JV company; *group members* are representatives of ”parent” companies in the board.

Dynamic Changes. As already described in the introduction a democratic group signature scheme has to handle the following dynamic events: join and leave of JV partners. Whenether a new ”parent” company signs the JV arrangement it provides its own representative to the board, i.e., a new member is added to the group. Similar, if a JV partner resigns from the contract its representative leaves the board, i.e., a current member is excluded from the group. In our model we assume that group members are notified about the occured dynamic changes.

The individual tracing property of democratic group signatures together with possible dynamic changes imply two additional (in the following informally defined) security requirements in the context of signer’s anonymity: (1) a joining member must not be able to open any group signature which has been produced by a group member before the join event took place, and (2) a leaving member must not be able to open any group signature which is produced by a group member after the exclusion process. More specific, only if users i and j are members during the period between two consecutive changes of the group formation, then member i can open all signatures that member j has generated within this period, and vice versa. Note that as a consequence of these requirements, all relevant group secrets that can be used to open group signatures have to be changed after every dynamic change of the group formation. A trivial solution is to reinitialize the group after any change of the group formation. Intuitively, this is inefficient because of additional interaction and computation costs. A more intelligent solution is to provide auxiliary protocols to hande join and leave events more efficiently, i.e., with less interaction and computation costs compared to a full reinitialization.

Trust Relationship. As mentioned in the introduction democratic group signatures are based on the ”trust, but verify” relationship between group members. Since tracing rights are given individually to every group member and anonymity is an issue, every group member is trusted not to reveal secrets that may be used to open group signatures to any other party. However, any group member may want to frame any other group member into signing a message, or generate a group signature that cannot be opened. This attack is imaginable in JV companies where partners may try to cheat and spend more money from the JVCo’s budget without refunding the difference to the budget. Additionally, some JV partners may try to collude against other JV partners and help each other to break the contract rules. In this case we assume that there is at least one group member who is honest. Obviously, this is a realistic assumption since a collusion which consists of all JV partners does not make any sense. Further, we assume that every member is able to authenticate own messages during the interaction with other members. This can be realized using public key certificates.

Counter. In order to simplify the handling of occuring dynamic events we distinguish between continuously changed group formations using a counter value t that consecutively counts occuring dynamic events. The counter is initialized as $t = 0$ after the initial group formation and increased by one after every further dynamic event. Thus, every value t corresponds to the group formation at the moment t has been set. Therefore, all parameters of our democratic group signature scheme are bound to a certain value t and may be changed with respect to the changes in the group formation.

2.2 Protocols and Algorithms

In this section we describe protocols and algorithms of a democratic group signature scheme. We denote by $Y_{[t]}$ the group public key, by $x_{i[t]}$ the secret signing key of member i, and by $\hat{x}_{[t]}$ the tracing trapdoor that correspond to a group formation identified by the counter value t, e.g., $Y_{[0]}$ and $Y_{[1]}$ denote the group public keys of the initial formation ($t = 0$) and after the first dynamic event ($t = 1$), respectively.

Definition 1. A democratic group signature scheme $\mathcal{DGS} = \{Setup(), Join(), Leave(), Sign(), Verify(), Trace(), VerifyTrace()\}$ is a digital signature scheme that consists of:

- A randomized protocol $Setup()$ between n cooperating users for the initial formation of the group. The public output is the group public key $Y_{[0]}$. The private outputs are the individual secret signing keys $x_{i[0]}$ for each member i, $i \in [1, n]$ and the tracing trapdoor $\hat{x}_{[0]}$.
- A randomized protocol $Join()$ between current group members and a joining member. Let t be a current counter value, n the number of current group members. The public output is the updated public key $Y_{[t+1]}$. The private outputs are possibly updated secret signing keys $x_{i[t+1]}$, $i \in [1, n + 1]$ for all members (including the new member) and the updated tracing trapdoor $\hat{x}_{[t+1]}$.
- A randomized protocol $Leave()$ between remaining group members. Let t be a current counter value, $n - 1$ the number of remaining group members. The public output is the updated public key $Y_{[t+1]}$. The private outputs are updated secret signing keys $x_{i[t+1]}$, $i \in [1, n - 1]$ for all remaining members and the updated tracing trapdoor $\hat{x}_{[t+1]}$.
- A randomized algorithm $Sign()$ that on input a secret signing key $x_{i[t]}$, a message m, and the group public key $Y_{[t]}$ outputs a signature σ.
- A deterministic algorithm $Verify()$ that on input a candidate signature σ, a message m, and the group public key $Y_{[t]}$ returns 1 if and only if σ was generated by a group member $i \in [1, n]$ using $Sign()$ on input $x_{i[t]}$, m and $Y_{[t]}$ for any counter value $t \in \mathbb{N}$.
- A randomized algorithm $Trace()$ that on input a candidate signature σ, a message m, the group public key $Y_{[t]}$, and the tracing trapdoor $\hat{x}_{[t]}$ returns the identity i of the group member who has generated σ together with a proof π of this fact for any counter value $t \in \mathbb{N}$.
- A deterministic algorithm $VerifyTrace()$ that on input a signature σ, a message m, the group public key $Y_{[t]}$, the identity i, and a candidate proof π that σ has been generated by member i returns 1 if and only if i and π were returned by $Trace()$ on input σ, m, $\hat{x}_{[t]}$, $Y_{[t]}$ for any counter value $t \in \mathbb{N}$.

We remark that $Join()$ is used by members to add new participants to the group, whereas $Leave()$ to exclude a certain member from the group according to the agreed membership policy.

Remark 1. Obviously, only the knowledge of the tracing trapdoor $\hat{x}_{[t]}$ allows to open produced group signatures. In order to fulfill the requirement of anonymity with respect to possible dynamic changes and in the context of the individual tracing rights, i.e., to prevent new members from opening previously generated group signatures, and former members from opening any further generated group signatures, the tracing trapdoor $\hat{x}_{[t]}$ has to be changed whenether t is increased after any occured dynamic event. Note that in Definition 1 the change of secret signing keys $x_{i[t]}$ is not explicitly required (i.e., we write "possibly updated" in the definition). Therefore, it depends on the concrete realization whether secret signing keys are changed or not.

2.3 Security Requirements

In this section we specify the security properties of a democratic group signature scheme.

Definition 2 (Correctness). *A democratic group signature scheme $\mathcal{DGS} = \{Setup(), Join(), Leave(), Sign(), Verify(), Trace(), VerifyTrace()\}$ is correct if for all $Y_{[t]}$, $x_{i[t]}$, and $\hat{x}_{[t]}$ returned by the protocols $Setup()$, $Join()$ and $Leave()$ with respect to the counter value t, and for any signature $\sigma = Sign(x_{i[t]}, m, Y_{[t]})$:*

$$Verify(\sigma, m, Y_{[t]}) = 1 \;\wedge\; Trace(\sigma, m, Y_{[t]}, \hat{x}_{[t]}) = (i, \pi) \;\wedge\; VerifyTrace(\sigma, m, Y_{[t]}, i, \pi) = 1.$$

In other words, the verification algorithm $Verify()$ accepts σ, and tracing algorithm $Trace()$ outputs i together with the proof π and verification algorithm $VerifyTrace()$ accepts this proof.

Traceability. We say that a democratic group signature scheme $\mathcal{DGS}$ is traceable if there exists no polynomial-time adversary A that can win the following traceability game, where A's goal is to forge a group signature that cannot be traced to one of the group members controlled or corrupted[1] by A. Our description of traceability includes collision-resistance, framing and unforgeability requirements as described in [4]. Let $\mathcal{A}_{[t]}$ denote a set of group members controlled by the adversary, and $\mathcal{A}'_{[t]}$ denote a set of group members corrupted by the adversary in the group formation identified by the counter value t, respectively.

> **Setup:** The challenger C performs the protocol $Setup()$ for n simulated participants and obtains the keys $Y_{[0]}$, $x_{i[0]}$ for all $i \in [1,n]$ and $\hat{x}_{[0]}$, and sets $t = 0$. It provides the adversary A with $Y_{[0]}$.
>
> **Queries:** After obtaining $Y_{[0]}$ adversary A can make the following queries:
> *Join.* A can initiate $Join()$ with C and introduce a new group member. Let t be the current counter value and n the current number of group members. The protocol updates the keys $Y_{[t+1]}$, $x_{i[t+1]}$ for all $i \in [1, n+1]$ and $\hat{x}_{[t+1]}$. A obtains $Y_{[t+1]}$, $\hat{x}_{[t+1]}$, the secret signing key $x_{a[t+1]}$ of the introduced member a. C adds the introduced member a to $\mathcal{A}_{[t+1]}$. Note that C does not learn $x_{a[t+1]}$.
>
> *Leave.* A can initiate $Leave()$ with C and exclude any member $i \in [1,n]$ from the group. Let t be the current counter value and n the current number of group members. The protocol updates the keys $Y_{[t+1]}$, $x_{i[t+1]}$ for all $i \in [1, n-1]$ and $\hat{x}_{[t+1]}$. C updates set $\mathcal{A}_{[t+1]}$ by removing the excluded member if he was in $\mathcal{A}_{[t]}$. If $\mathcal{A}_{[t+1]}$ is not empty, then A obtains $Y_{[t+1]}$, $\hat{x}_{[t+1]}$, and the secret signing keys $x_{a_i[t+1]}$ of all controlled members in $\mathcal{A}_{[t+1]}$; otherwise it obtains only $Y_{[t+1]}$.
>
> *CorruptMember.* A can request the secret signing key of any member $i \in [1,n]$ that is not controlled by A in a group formation identified by any $t \in \mathbb{N}$. C returns $x_{i[t]}$ to A, and adds member i to $\mathcal{A}'_{[t]}$.
>
> *CorruptGroupKey.* A can request the tracing trapdoor $\hat{x}_{[t]}$ for any $t \in \mathbb{N}$. C returns $\hat{x}_{[t]}$ to A.
>
> *Sign.* A can request a group signature of an arbitrary message m for any member $i \in [1,n]$ that is not controlled or corrupted by A and any counter value t. C computes and returns $\sigma = Sign(x_{i[t]}, m, Y_{[t]})$.
>
> **Output:** Finally, A returns a message m, a signature σ, and a counter value t. A wins if the forgery is successful, i.e., the following requirements are satisfied: (1) σ is accepted by the verification algorithm $Verify()$; (2) algorithm $Trace()$ traces σ to a group member that is neither in $\mathcal{A}_{[t]}$ nor in $\mathcal{A}'_{[t]}$, or fails; and (3) σ was not obtained by A from a signing query on m to C.

Definition 3. *A democratic group signature scheme $\mathcal{DGS} = \{Setup(), Join(), Leave(), Sign(), Verify(), Trace(), VerifyTrace()\}$ is traceable if the advantage of any polynomial-time adversary A in winning the traceability game defined as* $\mathbf{Adv}_A^{\text{tr}} = Pr[\text{A outputs a successful forgery}]$ *is negligible, i.e.,* $\mathbf{Adv}_A^{\text{tr}} \leq \epsilon$.

Anonymity. We say that a democratic group signature scheme $\mathcal{DGS}$ is anonymous if there exists no polynomial-time adversary A that can win the following anonymity game, where A's goal is to determine which of the two keys have been used to generate the signature. Let $\mathcal{A}_{[t]}$ denote a set of group members controlled by the adversary in the group formation identified by the counter value t, i.e., the adversary fully controls the participation of these members in the protocols of $\mathcal{DGS}$.

> **Setup:** The challenger C performs the protocol $Setup()$ for n simulated participants and obtains the keys $Y_{[0]}$, $x_{i[0]}$, $i \in [1,n]$ and $\hat{x}_{[0]}$. It provides the adversary A with $Y_{[0]}$.

[1] We distinguish between group members that are controlled and group members that are corrupted by A. If A introduces a new member to the group, then we say that A controls this member. If A obtains the secret signing key of a member that it has not introduced, then we say that this member is corrupted by A. Allowing A to control group members we consider an active adversary that participates in the protocols that update the group formation.

Type1-Queries: After obtaining $Y_{[0]}$ algorithm A can make the following queries:
Join. A can initiate $Join()$ with C and introduce a new group member. Let t be a current counter value and n the current number of group members. The protocol updates the keys $Y_{[t+1]}$, $x_{i[t+1]}$, $i \in [1, n+1]$ and $\hat{x}_{[t+1]}$. A obtains $Y_{[t+1]}$, $\hat{x}_{[t+1]}$, the secret signing key $x_{a[t+1]}$ of the introduced member a. C notes that there is a group member controlled by A in the group formation identified by $t+1$, thus adds the introduced member a to the set $\mathcal{A}_{[t+1]}$. Note that C does not learn $x_{a[t+1]}$.

Leave. A can initiate $Leave()$ with C and exclude any member $i \in [1, n]$ from the group. Let t be the current counter value and n the current number of group members. The protocol updates the keys $Y_{[t+1]}$, $x_{i[t+1]}$ for all $i \in [1, n-1]$ and $\hat{x}_{[t+1]}$. C updates set $\mathcal{A}_{[t+1]}$ by removing the excluded member if he was in $\mathcal{A}_{[t]}$. If $\mathcal{A}_{[t+1]}$ is not empty then A obtains $Y_{[t+1]}$, $\hat{x}_{[t+1]}$, and the secret signing keys $x_{a_i[t+1]}$ of all group members a_i in $\mathcal{A}_{[t+1]}$; otherwise it obtains only $Y_{[t+1]}$. Note that C does not learn any $x_{a_i[t+1]}$.

Sign. A can request a signature on an arbitrary message m for any member $i \in [1, n]$ that is not controlled by A and any counter value $t \in \mathbb{N}$. C computes $\sigma = Sign(x_{i[t]}, m, Y_{[t]})$ and returns σ to A.

Challenge: A outputs a message m', two identities of group members i_0, $i_1 \in [1, n]$ and a counter value t, such that $\mathcal{A}_{[t]} = \emptyset$. C chooses a random bit $d \in_R \{0, 1\}$, computes a signature $\sigma_d = Sign(x_{i_d[t]}, m', Y_{[t]})$ and returns it to A.

Type2-Queries: After obtaining the challenge, A can make the following queries:
Join. A can introduce new group members as in Type1-Queries.
Leave. A can exclude group members (also challenged members i_0 and i_1) as in Type1-Queries.
Sign. A can request a signature on an arbitrary message m (also m') for any member i that is not controlled by A (also for members i_0 and i_1) and any counter value $t \in \mathcal{N}$ as in Type1-Queries.

Output: Finally, A returns a bit d' trying to guess d, and wins the game if $d' = d$.

Definition 4. *A democratic group signature scheme $\mathcal{DGS} = \{Setup(), Join(), Leave(), Sign(), Verify(), Trace(), VerifyTrace()\}$ is anonymous if the advantage of any polynomial-time adversary A in winning the anonymity game defined as $\mathbf{Adv}_A^{\text{an}} = Pr[A(\sigma_1) = 1] - Pr[A(\sigma_0) = 1]$ is negligible, i.e., $\mathbf{Adv}_A^{\text{an}} \leq \epsilon$.*

We remark that at least two group members have to be in the group identified by the challenged counter value t, and $\mathcal{A}_{[t]} = \emptyset$ must hold. Note that the signature σ_d is bound to a certain counter value t so that the informally defined requirements from Section 2.1 (a joining member must not be able to open any group signature which has been produced by a group member before the join event took place, and a leaving member must not be able to open any group signature which is produced by a group member after the exclusion process) are covered by the above anonymity game.

Definition 5 (Security). *A democratic group signature scheme $\mathcal{DGS} = \{Setup(), Join(), Leave(), Sign(), Verify(), Trace(), VerifyTrace()\}$ is secure if it is correct, anonymous and traceable.*

3 Our Construction

3.1 Number-Theoretic Assumptions

Definition 6 (Discrete Logarithm (DL) Assumption). *Let $G = <g>$ be a cyclic group generated by g of order $ord(G)$. There is no probabilistic polynomial-time algorithm A that with non-negligible probability on input g^a where $a \in \mathbb{Z}_{ord(G)}$ outputs a. Let $\mathbf{Adv}_A^{\text{dl}} = Pr[A(g^a) = a]$ be the advantage of A in breaking the DL assumption. The DL assumption holds in G if this advantage is negligible, i.e, $\mathbf{Adv}_A^{\text{dl}} \leq \epsilon$.*

Definition 7 (Decisional Diffie-Hellman (DDH) Assumption). *Let $G =< g >$ be a cyclic group generated by g of order $ord(G)$. There is no probabilistic polynomial-time algorithm A that distinguishes with non-negligible probability between two distributions $D_0 = (g, g^a, g^b, g^c)$ and $D_1 = (g, g^a, g^b, g^{ab})$ where $a, b, c \in \mathbb{Z}_{ord(G)}$, i.e., A outputs 1 on input the distribution D_1, and 0 on input D_0. Let* $\mathbf{Adv}_A^{\text{ddh}} = Pr[A(D_1) = 1] - Pr[A(D_0) = 1]$ *be the advantage of A in breaking the DDH assumption. The DDH assumption holds in G if this advantage is negligible, i.e.,* $\mathbf{Adv}_A^{\text{ddh}} \leq \epsilon$.

3.2 Building Blocks

Our scheme consists of the following well known cryptographic primitives: a contributory group key agreement protocol for the initialization of the scheme and its maintenance upon dynamic changes, and signatures of knowledge for the signing and verification processes. Although our scheme may be seen as a straightforward solution, we note that to the best of our knowledge it is first (!) to consider contributory group key agreement protocols in the context of group-oriented signature schemes. This provides additional challenge for the security proof of the scheme, because the security of the interactive setup protocol has to be considered. Note that in classical group signatures the group manager that sets up the group is trusted, and, therefore, the security of the initialization procedure is usually omitted.

Contributory Group Key Agreement Protocols *Contributory group key agreement* (CGKA) protocols allow participants to form a group, compute the group secret key k_G by interaction, and update it on occured dynamic changes. The group secret key k_G is computed or updated as a function of individual contributions of all group members. CGKA protocols suit well into the scenario of the shared control based on the ”trust, but verify” relationship, because they are independent of any centralized management and allow group members to verify the protocol steps towards the computation of k_G. A CGKA protocol suite consists usually of a setup protocol and of protocols that handle various dynamic events, i.e., join and leave of single group members, and merge and partition of whole groups. However, for our group signature scheme we require only setup, join and leave protocols. Therefore, we omit the description of merge and partition protocols in the following definition.

Definition 8. A CGKA protocol suite $\mathcal{CGKA} = \{Setup(), (Join_i(), Join_u()), Leave()\}$ consists of the following algorithms and protocols:

- A randomized *interactive algorithm* $Setup()$ that implements the user's i side of the homonymous protocol between n users to initialize the group. On input an individual secret k_i of user i and corresponding contribution z_i, the algorithm obtains by interaction a set of individual contributions of other users $Z = \{z_j | j \in [1, n], j \neq i\}$, and outputs the group secret key k_G and some auxiliary information aux_i for the handling of further dynamic events. Note that k_G can be computed only by group members who participate in the protocol.
- A pair of randomized *interactive algorithms* $(Join_i(), Join_u())$ that implement member's i and joining user's u sides of the protocol $Join()$ between n group members and the joining user, respectively.
  $Join_i()$ takes as input a current individual secret k_i and current information aux_i, obtains by interaction the joining user's contribution z_u, and outputs updated k_G, aux_i, and possibly changed k_i.
  $Join_u()$ takes as input a new user's individual secret k_u and corresponding contribution z_u, obtains by interaction some auxiliary information aux_u (including set $Z = \{z_j | j \in [1, n]$), and outputs updated k_G and aux_u.
- A randomized *interactive algorithm* $Leave()$ that implements member's i side of the homonymous protocol between remaining $n - 1$ group members due to exclusion of a member j. On input a current individual secret k_i, leaving member's contribution z_j and current information aux_i, the algorithm outputs updated k_G, aux_i, and possibly changed k_i.

Remark 2. CGKA protocols that handle dynamic events require usually that some members change their individual secret keys k_i during the protocol for the sake of security, e.g., to guarantee the freshness of the updated group secret key. This is emphasized by the expression ”possibly changed” in the above definition of the protocols $Join_i()$ and $Leave()$. If such change is required then member i changes k_i and updates own contribution z_i. Thus, no other party except for member i ever learns k_i.

Remark 3. The auxiliary information aux_i returned by the interactive algorithms of $\mathcal{CGKA}$ depends on the actual protocol suite. It contains auxiliary values that can be used by group members to handle further occuring dynamic events. We remark that for our scheme aux_i must provide every member i with a current set of members' contributions $Z = \{z_j | j \in [1, n]\}$, because we use Z as part of the group public key. This requirement is implicitly achieved for most CGKA protocols (including those mentioned below), because participants broadcast contributions over a public channel. This is also the reason why considering contributions as part of the group public key is not a hazard to the security of the CGKA protocols.

The security of CGKA protocol suites is usually described by the following (informally described) set of requirements ([17]):

– *Computational group key secrecy* requires that for a passive adversary it must be computationally infeasible to discover any secret group key.
– *Decisional group key secrecy*[2] requires that for a passive adversary it must be computationally infeasible to distinguish any bits of the secret group key from random bits.
– *Forward secrecy*[3] requires that any passive adversary being in possession of a subset of old group keys must not be able to discover any subsequent group key (e.g. if a member leaves the group knowing the group key it should not be able to compute the updated group key).
– *Backward secrecy* requires that any passive adversary being in possession of a subset of contiguous group keys must not be able to discover any preceding group key (e.g. if a member joins to the group and learns the updated group key it should not be able to compute the previous group key).
– *Key independence* requires that any passive adversary being in possession of any subset of group keys must not be able to discover any other group key.

Note that the adversary is assumed to be passive, i.e., it is not a valid group member during the time period the attack is taking place. Considering active adversaries, i.e., group members does not make sense, because every group member learns the group key by the end of the protocol. Thus, group members are trusted not to reveal the group key or any other secret values that may lead to the computation of the group key to the third parties. This is exactly the requirement stated for democratic group signatures in Section 2.1 to keep the individual tracing of signatures prior to group members. Further reason for the suitability of CGKA protocols for the initialization and maintenance of the DGS is the ”trust, but verify” relationship between group members, i.e., some protocols, like [16] and [17] define a role of a *sponsor* that becomes active on dynamic events and performs some computations on behalf of the group to achieve additional efficiency. Although this sponsor acts on behalf of the group, there exists at least one other group member who can verify the sponsor's actions. Additionally, join and leave protocols can be utilized in DGS to maintain the scheme efficiently, i.e., without reinitialization.

Members' contributions in the mentioned CGKA protocols are constructed as follows: every member chooses own secret k_i and computes his contribution $z_i = g^{k_i}$, where g is a generator of a cyclic group $G =< g >$, where the DL Assumption holds (e.g., $\mathbb{Z}_p^*$ with prime p or a subgroup of points on an elliptic curve E over a finite field $\mathbb{F}_q$).

Our scheme has been designed to work with any CGKA protocol suite that fulfills described security requirements, and the contributions of group members have the above construction. For example, protocol suites in [16] and [17] can be applied in the scheme. In the following we briefly explain how we use the properties of such CGKA protocols.

The group secret key k_G computed and updated by the CGKA protocols is used as the tracing trapdoor $\hat{x}$ from Definition 1. In order to maintain $\hat{x}$ after dynamic changes and keep individual tracing rights prior to group members, the protocols $Setup()$, $(Join_i()$, $Join_u())$ and $Leave()$ of the CGKA protocol suite are embedded in the homonymous protocols of our DGS, respectively. Security requirements of CGKA

[2] The formal definition of the decisional group key secrecy is given in Appendix A.3 where it is used to prove the anonymity of our scheme.
[3] In the context of our security proof (Remark 5) we show that the absence of so-called Perfect Forward Secrecy does not provide additional risks to the security of our scheme.

protocols take care that no unauthorized users are able to compute $\hat{x}$ and open group signatures. In our scheme members use their individual secrets k_i as secret signing keys x_i from Definition 1. This is possible, because every k_i remains known only to the corresponding member. Our scheme includes the set of members' contributions $Z = \{z_i | \forall i \in [1, n]\}$ in the group public key Y as shown in Section 3.3.

Signatures of Knowledge *Signatures of knowledge*, introduced in [11], and also used in some classical group signature schemes, like [10] and [3] are message dependent zero-knowledge proofs of knowledge of some secret s that are made non-interactive using the Fiat-Shamir heuristic [14]. The security of such schemes is usually shown by proving the security of an underlying interactive zero-knowledge protocol and then by assuming that no security flaws occur if verifier's computations in the interacive protocol are replaced by a collision resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^k$ with security parameter k. The security of this non-interactive approach can be shown in the random oracle model [5].

Signatures of knowledge consist of two polynomial-time algorithms ($SKSig()$, $SKVer()$), where $SKSig()$ is a randomized signing algorithm and $SKVer()$ a deterministic verifying algorithm. A signer *S* who is in possession of some secret s can compute the signature of knowledge of s on a message m using $SKSig()$ and send it to a verifier *V*. In $SKSig()$ an appropriate one-way function f is applied to the secret that prevents any leakage of the information about the secret. If algorithm $SKVer()$ performed by *V* accepts the signature then *V* is convinced that *S* knows s, but learns nothing about this secret. However, if *S* does not know s or does not use s in $SKSig()$ to compute the signature then $SKVer()$ rejects. A signature of knowledge is called secure if the probability of producing a forged signature without knowing the secret s such that $SKVer()$ accepts the forgery is negligible, and if any correctly generated signature does not reveal any sufficient information that may be used to compute s. Before we describe the signatures of knowledge used in our scheme, we give a simple example to explain used notations, borrowed from [11]: a signature of knowledge on a message m, denoted $SK[(\alpha) : y = g^{\alpha}](m)$, proves the knowledge of the discrete logarithm of y to the base g as described by the equation on the right side of the colon. By the convention Greek letters denote secret values, whose knowledge has to be proved, whereas other letters denote public values.

The following is a combination of the signature of knowledge of the representation and knowledge of 1-out-of-n discrete logarithms. We extend the signature of knowledge of 1-out-of-2 discrete logarithms from [10]. Note that the signature does not reveal which discrete logarithm the signer knows.

Definition 9. *Let G be a cyclic group of order $ord(G)$ where the Discrete Logarithm Assumption holds. A $3n$-tuple $(c_1, \ldots, c_n, s_{u_1}, \ldots, s_{u_n}, s_{v_1}, \ldots, s_{v_n}) \in (\{0,1\}^k)^n \times \mathbb{Z}^{2n}_{ord(G)}$ satisfying $c_1 \oplus \ldots \oplus c_n = H(g_1, g_2, y, z_1, \ldots, z_n, y^{c_1} g_1^{s_{u_1}} g_2^{s_{v_1}}, \ldots, y^{c_n} g_1^{s_{u_n}} g_2^{s_{v_n}}, z_1^{c_1} g_2^{s_{v_1}}, \ldots, z_n^{c_n} g_2^{s_{v_n}}, m)$ is a signature of knowledge of the discrete logarithm of one z_i, $i \in [1, n]$ to the base g_2 and its equality to the exponent of the g_2-part of y, on a message $m \in \{0,1\}^*$, denoted*

$$SK[(\alpha_i, \beta) : y = g_1^{\beta} g_2^{\alpha_i} \wedge (z_1 = g_2^{\alpha_1} \vee \ldots \vee z_n = g_2^{\alpha_n})](m).$$

Assume that the signer knows $(x_1, x_2) \in_R \mathbb{Z}^2_{ord(G)}$ with $y = g_1^{x_1} g_2^{x_2}$ and $z_i = g_2^{x_2}$ for one $i \in [1, n]$. Then a signature $SK[(\alpha_i, \beta) : y = g_1^{\beta} g_2^{\alpha_i} \wedge (z_1 = g_2^{\alpha_1} \vee \ldots \vee z_n = g_2^{\alpha_n})](m)$ on a message $m \in \{0,1\}^*$ can be computed using the following algorithm:

- $(r_{u_1}, \ldots, r_{u_n}, r_{v_1}, \ldots, r_{v_n}) \in_R \mathbb{Z}^{2n}_{ord(G)}$; $(c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n) \in_R (\{0,1\}^k)^{n-1}$;
- $u_i = g_1^{r_{u_i}}$; $v_i = g_2^{r_{v_i}}$; for all $j \neq i$: $u_j = y^{c_j} g_1^{r_{u_j}} g_2^{r_{v_j}}$ and $v_j = z_j^{c_j} g_2^{r_{v_j}}$;
- $c_i = \bigoplus_{j \neq i}^{n} c_j \oplus H(g_1, g_2, y, u_1, \ldots, u_{i-1}, u_i v_i, u_{i+1}, \ldots, u_n, v_1, \ldots, v_n, m)$;
- $s_{u_i} = r_{u_i} - c_i x_1$; $s_{v_i} = r_{v_i} - c_i x_2$; for all $j \neq i$: $s_{u_j} = r_{u_j}$ and $s_{v_j} = r_{v_j}$.

The following is a combination of the signatures of knowledge of two equal discrete logarithms and of the representation described in [11] and [10].

Definition 10. *Let G be a cyclic group of order $ord(G)$ where the Discrete Logarithm Assumption holds. A tupel $(c, s_1, s_2) \in \{0,1\}^k \times \mathbb{Z}^2_{ord(G)}$ satisfying $c = H(g_1, g_2, y_1, y_2, y_1^c g_2^{s_{w_1}}, y_2^c g_1^{s_{w_1}} g_2^{s_{w_2}}, m)$ is a signature of knowledge of the representation of y_2 to the bases g_1 and g_2, and of the equality of the discrete logarithm of y_1 to the base g_2 and the exponent of the g_1-part of y_2, on a message $m \in \{0,1\}^*$, denoted*

$$SK[(\alpha, \beta) : y_1 = g_2^{\beta} \wedge y_2 = g_1^{\beta} g_2^{\alpha}](m).$$

Assume the signer knows $(x_1, x_2) \in \mathbb{Z}^2_{ord(G)}$ with $y_1 = g_2^{x_1}$ and $y_2 = g_1^{x_1} g_2^{x_2}$. Then a signature $SK[(\alpha, \beta) : y_1 = g_2^{\beta} \wedge y_2 = g_1^{\beta} g_2^{\alpha}](m)$ on a message $m \in \{0,1\}^*$ can be computed using the following algorithm:

- $(r_{w_1}, r_{w_2}) \in_R \mathbb{Z}^2_{ord(G)}$; $w_1 = g_2^{r_{w_1}}$; $w_2 = g_1^{r_{w_1}} g_2^{r_{w_2}}$;
- $c = H(g_1, g_2, y_1, y_2, w_1, w_2, m)$; $s_{w_1} = r_{w_1} - cx_1$; $s_{w_2} = r_{w_2} - cx_2$.

Both schemes can be proven secure in the random oracle model.[4] We remark that the interactive version of these protocols are zero-knowledge. Furthermore, these signatures of knowledge can be combined into a single signature of knowledge as shown in the signing protocol in Section 3.3. Signatures of knowledge can also be used as non-interactive proofs of knowledge if an empty string is used instead of the message m, i.e., $SK[(\alpha) : y = g^{\alpha}]()$ stands for the proof of knowledge of a discrete logarithm of y to the base g.

3.3 The Scheme

In this section we describe in detail our democratic group signature scheme with respect to Definition 1. Consider a cyclic group $G = < g >$ of order $ord(G)$ generated by g where the Discrete Logarithm Assumption holds. Let $\mathcal{CGKA} = \{Setup(), (Join_i(), Join_u()), Leave()\}$ be a secure contributory group key agreement protocol suite where group members' contributions are constructed as described in Section 3.2. We assume that all sent messages are authentic.

Protocol $Setup()$**.** The protocol between n users to set up a SDG proceeds as follows. Each user i:

- sets counter value $t = 0$, selects secret signing key $x_{i[0]} \in_R \mathbb{Z}_{ord(G)}$, computes corresponding contribution $z_{i[0]} = g^{x_{i[0]}}$,
- performs the instance of the interactive algorithm $\mathcal{CGKA}.Setup(x_{i[0]}, z_{i[0]})$, and obtains the tracing trapdoor (group secret key) $\hat{x}_{[0]} \in \mathbb{Z}_{ord(G)}$ and auxiliary information $aux_{i[0]}$ that contains the set of contributions of all group members, i.e., $Z_{[0]} = \{z_{1[0]}, \ldots, z_{n[0]}\}$,
- computes $\hat{y}_{[0]} = g^{\hat{x}_{[0]}}$, and defines the group public key $Y_{[0]} = (\hat{y}_{[0]}, Z_{[0]})$.

The public output of the protocol is the group public key $Y_{[0]}$. The private outputs are $x_{i[0]}$, $\hat{x}_{[0]}$ and $aux_{i[0]}$.

Publishing of $Y_{[t]}$. Every group public key $Y_{[t]} = \{\hat{y}_{[t]}, Z_{[t]}\}$ has to be published in an authentic manner. Unlike in classical group signatures, where the group manager usually proves the correctness of the group public key to the certification authority, in DGS all group members have to cooperate for this purpose. We suggest the following simple solution. Every participant i holds an identity certificate $cert_i$ on a public key pk_i issued by a certification authority CA and used to authenticate messages of i. The corresponding private key sk_i is known only to the participant. Participant i computes the following signature of knowledge on a message $M_i = cert_i || t || Y_{[t]}$:

$$S_i = SK[(\alpha_i, \beta) : \hat{y}_{[t]} = g^{\beta} \wedge z_{i[t]} = g^{\alpha_i}](M_i),$$

and signs it using a digital signature scheme with his private key sk_i, i.e., $T_i = Sign(sk_i, S_i)$. Then, every member publishes (M_i, S_i, T_i). Every member verifies whether all published tuples are correct, i.e., all proofs are verifiable, and every message M_i contains the same set of contributions $Z_{[t]}$ in $Y_{[t]}$, and complains if he discovers any cheating attempt. Obviously, the signature S_i proves that member i knows the tracing trapdoor $\hat{x}_{[t]}$, own secret $x_{i[t]}$ used to compute the contribution $z_{i[t]}$ and seals the group formation identified by the counter value t, because contributions of all group members are part of M_i. This ensures that any member's attempt to cheat will be discovered if there is at least one honest group member (as required by the trust relationship in Section 2.1). The cheating member can be identified using his signature T_i. The published signature T_i can then be used to identify a group member i upon his contribution $z_{i[t]}$.

Remark 4. This procedure has to be performed whenether the group public key $Y_{[t]}$ is updated, i.e., after the protocols $Join()$ and $Leave()$. Since $Y_{[t]}$ changes over dynamic events group signatures produced by our scheme can be used by members only to prove their membership in the group formation identified by t. To prove the membership in several formations a signature for each formation must be provided.

[4] Note that not all protocols, which can be proven to be secure in the random oracle model are also secure in the standard model as recently shown in [12]. However, it is still believed that for the kind of protocols considered here (i.e., signatures of knowledge) random oracle provides sufficient proofs.

Protocol $Join()$**.** The protocol between the group and a joining member u proceeds as follows. Let t be the current counter value and n the number of group members.

- Joining member u selects his secret signing key $x_{u[t+1]} \in_R \mathbb{Z}_{ord(G)}$, computes corresponding contribution $z_{u[t+1]} = g^{x_{u[t]}}$.
- Group members and the joining member perform the protocol $\mathcal{CGKA}.Join()$ by calling the instances $Join_i()$ and $Join_u()$, respectively:
  $\mathcal{CGKA}.Join_i(x_{i[t]}, aux_{i[t]})$ is called by every member $i \in [1, n]$, and outputs the updated $\hat{x}_{[t+1]}$, $aux_{i[t+1]}$, and possibly updated $x_{i[t+1]}$.
  $\mathcal{CGKA}.Join_u(x_{u[t+1]}, z_{u[t+1]})$ is called by the joining member u, and outputs the updated $\hat{x}_{[t+1]}$ and $aux_{u[t+1]}$.
- Every group member increases t, computes $\hat{y}_{[t+1]} = g^{\hat{x}_{[t+1]}}$ and $Y_{[t+1]} = (\hat{y}_{[t+1]}, z_{1[t+1]}, \ldots, z_{n+1[t+1]})$.

The public output of the protocol is the changed group public key $Y_{[t+1]}$. The private outputs are $x_{i[t+1]}$, $\hat{x}_{[t+1]}$, and $aux_{i[t+1]}$. We remark, that current value t can be sent to the joining member as part of $aux_{u[t+1]}$. This requires a minimal modification of the underlying CGKA protocol. If no modification is possible, then one additional message containing t has to be sent between the group and the joining member.

Protocol $Leave()$**.** The protocol between the remaining group members after a member u has left the group proceeds as follows. Let t be the current counter value and n the number of group members. Every remaining group member $i \in [1, n-1]$:

- performs the instance of the interactive algorithm $\mathcal{CGKA}.Leave(x_{i[t]}, z_{u[t]}, aux_{i[t]})$, and obtains the updated $\hat{x}_{[t+1]}$, $aux_{i[t+1]}$, and possibly updated $x_{i[t+1]}$;
- increases t, computes $\hat{y}_{[t+1]} = g^{\hat{x}_{[t+1]}}$ and $Y_{[t+1]} = (\hat{y}_{[t+1]}, z_{1[t+1]}, \ldots, z_{n-1[t+1]})$.

The public output of the protocol is the updated group public key $Y_{[t+1]}$. The private outputs are $x_{i[t+1]}$, $\hat{x}_{[t+1]}$, and $aux_{i[t+1]}$.

Algorithm $Sign()$**.** The signing algorithm is a combination of the signatures of knowledge from Section 3.2. In order to generate a group signature on a message $m \in \{0,1\}^*$ the algorithm on input $x_{i[t]}$, m and $Y_{[t]} = (\hat{y}_{[t]}, Z_{[t]})$ performs the following computations:

- $r \in_R \mathbb{Z}_{ord(G)}$; $\tilde{g} = g^r$; $\tilde{y} = \hat{y}_{[t]}^r z_{i[t]}$;
- $S = SK[(\alpha_i, \beta) : \tilde{g} = g^\beta \wedge \tilde{y} = \hat{y}_{[t]}^\beta g^{\alpha_i} \wedge (z_{1[t]} = g^{\alpha_1} \vee \ldots \vee z_{n[t]} = g^{\alpha_n})](m)$ is computed as follows:
  * $(r_{u_1}, \ldots, r_{u_n}, r_{v_1}, \ldots, r_{v_n},) \in_R \mathbb{Z}_{ord(G)}^{2n}$; $(c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n) \in_R (\{0,1\}^k)^{n-1}$;
  * $u_i = \hat{y}_{[t]}^{r_{u_i}}$; $v_i = g^{r_{v_i}}$; $w_i = g^{r_{u_i}}$; for all $j \neq i$: $u_j = \tilde{y}^{c_j} \hat{y}_{[t]}^{r_{u_j}} g^{r_{v_j}}$, $v_j = z_{j[t]}^{c_j} g^{r_{v_j}}$; $w_j = \tilde{g}^{c_j} g^{r_{u_j}}$;
  * $c_i = \bigoplus_{j \neq i}^{n} c_j \oplus H(g, \hat{y}_{[t]}, \tilde{g}, \tilde{y}, u_1, \ldots, u_{i-1}, u_i v_i, u_{i+1}, \ldots, u_n, v_1, \ldots, v_n, w_1, \ldots, w_n, m)$;
  * $s_{u_i} = r_{u_i} - c_i r$; $s_{v_i} = r_{v_i} - c_i x_{i[t]}$; for all $j \neq i$: $s_{u_j} = r_{u_j}$, $s_{v_j} = r_{v_j}$;
  * $S = (c_1, \ldots, c_n, s_{u_1}, \ldots, s_{u_n}, s_{v_1}, \ldots, s_{v_n})$;

The algorithm outputs the group signature $\sigma = (\tilde{g}, \tilde{y}, S)$. Obviously, the signer i proves that values $(\tilde{g}, \tilde{y})$ encrypt his contribution $z_{i[t]}$ without revealing the latter by proving the knowledge of the representation of $\tilde{y}$ to the bases $\hat{y}_{[t]}$ and g, and that the same exponent is used to compute $\tilde{g}$ and the $\hat{y}_{[t]}$-part in $\tilde{y}$. That encrypted value is a valid contribution in $Z_{[t]}$ is shown by proving the equality between the exponent in g-part of $\tilde{y}$ and the discrete logarithm of $z_{i[t]}$ to the base g. The fact that the signer knows the discrete logarithm that has been used to compute $z_{i[t]}$ proves that the signer is the owner of this contribution. Note that the signature does not reveal the contribution of the signer. This is important for the anonymity property of the scheme.

Algorithm $Verify()$**.** The verifying algorithm on input a candidate group signature $\sigma = (\tilde{g}, \tilde{y}, S)$, message m, and the group public key $Y_{[t]} = (\hat{y}_{[t]}, Z_{[t]})$ verifies S, i.e.,

- Parse S as $(c_1, \ldots, c_n, s_{u_1}, \ldots, s_{u_n}, s_{v_1}, \ldots, s_{v_n})$; $\bar{c} = \bigoplus_{j=1}^{n} c_j$;
- check $\bar{c} \overset{?}{=} H(g, \hat{y}_{[t]}, \tilde{g}, \tilde{y}, \tilde{y}^{c_1} \hat{y}_{[t]}^{s_{u_1}} g^{s_{v_1}}, \ldots, \tilde{y}^{c_n} \hat{y}_{[t]}^{s_{u_n}} g^{s_{v_n}}, z_{1[t]}^{c_1} g^{s_{v_1}}, \ldots, z_{n[t]}^{c_n} g^{s_{v_n}}, \tilde{g}^{c_1} g^{s_{u_1}}, \ldots, \tilde{g}^{c_n} g^{s_{u_n}}, m)$

and returns 1 if verification is successful; otherwise the algorithm fails.

Algorithm $Trace()$**.** The tracing algorithm on input a candidate signature $\sigma = (\tilde{g}, \tilde{y}, S)$, a message m, the group public key $Y_{[t]} = (\hat{y}_{[t]}, Z_{[t]})$, and the tracing trapdoor $\hat{x}_{[t]}$ proceeds as follows:

- check $Verify(\sigma, m, Y_{[t]}) \overset{?}{=} 1$;
- $V_1 = \tilde{g}^{\hat{x}_{[t]}}$; $z_{i[t]} = \tilde{y}/V_1$;
- check $z_{i[t]} \overset{?}{\in} Y_{[t]}$;
- $V_2 = SK[(\alpha) : \hat{y}_{[t]} = g^{\alpha} \wedge V_1 = \tilde{g}^{\alpha}]()$

It outputs the decrypted signer's contribution $z_{i[t]}$ and a proof (V_1, V_2) if and only if all checks are successful; otherwise the algorithm fails. V_2 is a proof that V_1 equals to the $\hat{y}_{[t]}$-part of $\tilde{y}$, i.e., $V_1 = \hat{y}_{[t]}^{r}$. The signer's identity can be easily computed from his contribution $z_{i[t]}$ as all contributions are part of the group public key $Y_{[t]}$ and have been signed by the corresponding member using his certified key pair (sk_i, pk_i).

Algorithm $VerifyTrace()$**.** The algorithm on input a signature $\sigma = (\tilde{g}, \tilde{y}, S)$, a message m, the group public key $Y_{[t]} = (\hat{y}_{[t]}, Z_{[t]})$, the contribution $z_{i[t]}$, and proving values (V_1, V_2) proceeds as follows:

- check $Verify(\sigma, m, Y_{[t]}) \overset{?}{=} 1$;
- check $\tilde{y} \overset{?}{=} V_1 z_{i[t]}$;
- verify V_2;

It outputs 1 if and only if all checks and verifications are successful; otherwise the algorithm fails.

3.4 Security Analysis

In this section we analyze the security of our scheme with respect to the requirements in Section 2.3. We show that as long as underlying cryptographic building blocks (i.e, CGKA protocol and signatures of knowledge) are secure our scheme fulfills the stated requirements. Since signatures of knowledge are provably secure only in the random oracle model we apply this model to prove the security of our scheme.

Lemma 1. *The construction of a democratic group signature scheme $\mathcal{DGS} = \{Setup(), Join(), Leave(), Sign(), Verify(), Trace(), VerifyTrace()\}$ from Section 3.3 is correct.*

Proof. The full proof is presented in Appendix A.1.

Lemma 2. *The democratic group signature scheme $\mathcal{DGS}$ from Section 3.3 is traceable in the random oracle model assuming that contributory group key agreement protocol suite $\mathcal{CGKA}$ is secure and the Discrete Logarithm (DL) assumption holds in the group $G = <g>$.*

Proof. The full proof is presented in Appendix A.2.

Lemma 3. *The democratic group signature scheme $\mathcal{DGS}$ from Section 3.3 is anonymous in the random oracle model assuming that contributory group key agreement protocol suite $\mathcal{CGKA}$ is secure and the Decisional Diffie-Hellman (DDH) assumption holds in the group $G = <g>$.*

Proof. The full proof is presented in Appendix A.3.

Theorem 1. *The democratic group signature scheme $\mathcal{DGS}$ from Section 3.3 is secure in the random oracle model assuming that contributory group key agreement protocol suite $\mathcal{CGKA}$ is secure, and the DDH and DL assumptions hold in the group $G = <g>$.*

Proof. The proof of this theorem follows immediately from Definition 5 and Lemmas 1, 2, and 3. □

4 Conclusion and Further Directions

In order to handle the shared financial control in JVCos based on the ”trust, but verify” relationship we have proposed a new group-oriented signature scheme, called a *democratic group signature*. We have shown that the classical model of group signatures is not applicable for this scenario because of the group manager’s role, which contradicts to the requirement of individual tracing of group signatures and joint control over the membership in the group. Our model can also be used for other issues of the shared control where the ”trust, but verify” relationship is required. The proposed practical construction fulfills the stated requirements and is secure in the random oracle model. The signature of knowledge used in the signing algorithm is of independent interest. It may be used to prove that $(\tilde{g}, \tilde{y})$ encrypts a value g^{α_i} from the set $g^{\alpha_1}, \ldots, g^{\alpha_n}$ without revealing the index i. In the following we would like to point out some further research topics in the area of democratic group signatures.

4.1 Dynamic Changes

The presented scheme uses a contributory group key agreement protocol suite to establish and maintain the group formation upon dynamic changes. The scheme is currently designed to work with any protocol suite which fulfills the requirements from Section 3.2. However, not all CGKA protocols provide same efficiency in this context. It would be interesting to analyze, which of the existing CGKA protocol suites are mostly efficient? Further details and comparison of existing CGKA protocol suites are given in [2].

4.2 Signature and Public Key Sizes

In the proposed practical solution the group public key and generated signatures grow linearly in the number of current group members, whereas recent classical group signature schemes keep both sizes constant. The latter is possible because the group manager controls the group formation and maintains a database with transcripts of join events for all members. This database is used to open group signatures and its size grows linearly (!) in the number of current group members. A challenging task is to design a DGS which fulfills the proposed requirements, but keeps the signature size and (or) the group public key size constant.

References

1. M. Abe, M. Ohkubo, and K. Suzuki. 1-out-of-n signatures from a variety of keys. In *ASIACRYPT ’02: Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security*, volume 2501 of *Lecture Notes in Computer Science*, pages 415–432. Springer-Verlag, 2002.
2. Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik. On the performance of group key agreement protocols. *ACM Trans. Inf. Syst. Secur.*, 7(3):457–488, 2004.
3. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO ’00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer-Verlag, 2000.
4. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *Advances in Cryptology (EUROCRYPT 2003),Lecture Notes in Computer Science*, volume 2656, pages 614–629. Springer-Verlag, 2003.
5. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM Press, 1993.
6. M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA, Lecture Notes in Computer Science*, volume 2656, pages 136–153. Springer-Verlag, 2005.
7. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Advances in Cryptology—CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer-Verlag, 2004.
8. J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In *SCN, Lecture Notes in Computer Science*, volume 3352, pages 120–133. Springer-Verlag, 2004.
9. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology—CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer-Verlag, 2004.
10. J. Camenisch and M. Michels. A group signature scheme with improved efficiency. In *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, pages 160–174. Springer-Verlag, 1998.
11. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology, Lecture Notes in Computer Science*, volume 1294, pages 410–424. Springer-Verlag, 1997.
12. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *30th Annual ACM Symposium on Theory of Computing (STOC)*, 1998.

13. D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology (EUROCRYPT 1991),Lecture Notes in Computer Science*, volume 547, pages 257–265. Springer-Verlag, 1991.
14. A. Fiat and A. Shamir. How to prove yourself: Practical solution to identification and signature problems. In *Advances in Cryptology - CRYPTO 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987.
15. A. Kiayias and M. Yung. Extracting group signatures from traitor tracing schemes. In *Advances in Cryptology (EUROCRYPT 2003),Lecture Notes in Computer Science*, volume 2656, pages 630–648. Springer-Verlag, 2003.
16. Y. Kim, A. Perrig, and G. Tsudik. Communication-efficient group key agreement. In *Information Systems Security, Proc. of the 17th International Information Security Conference, IFIP SEC'01*, 2001.
17. Y. Kim, A. Perrig, and G. Tsudik. Tree-based group key agreement. *ACM Transactions on Information and System Security*, 7(1):60–96, 2004.
18. J. K. Liu, V. K. Wei, and D. S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In *Information Security and Privacy: 9th Australasian Conference, ACISP 2004.*, volume 3108 of *Lecture Notes in Computer Science*, pages 325–335. Springer-Verlag, 2004.
19. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(3):361–396, 2000.
20. R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer-Verlag, 2001.
21. C. zhi Gao, Z. an Yao, and L. Li. A ring signature scheme based on the Nyberg-Rueppel signature scheme. In *Applied Cryptography and Network Security, First International Conference, ACNS 2003.*, volume 2846 of *Lecture Notes in Computer Science*, pages 169–175. Springer-Verlag, 2003.

A Security Proofs

A.1 Proof of Lemma 1 (Correctness)

To prove the correctness of the scheme we have to show according to Definition 2 that for all $Y_{[t]}$, $x_{i[t]}$ with $i \in [1, n]$, and $\hat{x}_{[t]}$ if $\sigma = Sign(x_{i[t]}, m, Y_{[t]})$ then the following statements hold:

(i) $Verify(\sigma, m, Y_{[t]}) = 1$ and
(ii) $Trace(\sigma, m, Y_{[t]}, \hat{x}_{[t]}) = (z_{i[t]}, (V_1, V_2))$ and $VerifyTrace(\sigma, m, Y_{[t]}, z_{i[t]}, (V_1, V_2)) = 1$.

Let $\sigma = (\tilde{g}, \tilde{y}, S)$ and $Y_{[t]} = (\hat{y}_{[t]}, z_{1[t]}, \ldots, z_{n[t]})$. The statement $Verify(\sigma, m, Y_{[t]}) = 1$ holds since any correctly generated signature of knowledge S is accepted by the corresponding verification algorithm. On input σ, m, $Y_{[t]}$, and $\hat{x}_{[t]}$ the tracing algorithm computes $V_1 = \tilde{g}^{\hat{x}_{[t]}}$, which is equivalent to $\hat{y}^r_{[t]}$, therefore $\tilde{y}/V_1 = \hat{y}^r_{[t]} z_{i[t]}/\hat{y}^r_{[t]} = z_{i[t]}$ holds. The contribution $z_{i[t]}$ is part of $Y_{[t]}$ and reveals the identity of the signer. The tracing algorithm outputs also the proof of knowledge V_2, which is constructed using the secret group key $\hat{x}_{[t]}$. Algorithm $VerifyTrace()$ uses the proof (V_1, V_2) to verify the correctness of $z_{i[t]}$ returned by the tracing algorithm with the equation $\tilde{y} \overset{?}{=} V_1 z_{i[t]}$, which obviously holds, because $V_1 z_{i[t]} = \hat{y}^r_{[t]} z_{i[t]} = \tilde{y}$. Verification of V_2 is succesfull due to its construction by the tracing algorithm. Thus, $VerifyTrace(\sigma, m, Y_{[t]}, z_{i[t]}, (V_1, V_2)) = 1$. Hence, the democratic group signature scheme is correct. □

A.2 Proof of Lemma 2 (Traceability)

The proof of traceability consists of four parts. First, we describe an oracle K that is used by the polynomial-time challenger B to perform $CGKA$ protocols. Second, we describe the interaction between B and polynomial-time adversary A that is assumed to win the traceability game for the group signature scheme. Third, we distinguish between different forgery types possibly produced by A. Fourth, we show how interaction between B and A can be modified to compute a discrete logarithm in the group $G = <g>$ using the forking lemma [19, Theorem 3].

Oracle K**.** In the following interaction we outsource the challenger's part of the contributory group key protocol $CGKA$ used in $\mathcal{DGS}$ to an oracle K. The idea is that B uses K to initialize the group of n members and acts as a mediator between A and K during the interaction with A. Since K initializes the group all secret keys $\{x_i | i \in [1, n]\}$ are initially known to K, but not to B. If during the interaction A

queries B to perform a dynamic change (join or leave) then B asks K to proceed with the query. Using K we allow indirect participation of B in the $CGKA$ protocols. B is allowed to ask for the computed tracing trapdoor $\hat{x}$ and any secret signing key $\{x_i | i \in [1, n]\}$, and receives also contributions of all group members, i.e., set $Z = \{z_i = g^{x_i} | i \in [1, n]\}$. Our goal is to use the interaction to compute a secret signing key x_j that remains unknown to B after the interaction is finished. This is equivalent to the computation of the discrete logarithm in G since $x_j = \log_g z_j$.

Let $\mathcal{K}$ denote a set of contributions of group members that are controlled by the oracle K, and n be the current number of group members. Oracle K answers to the following queries of B:

Setup. This query can be used to initialize the group. It contains $n \in \mathbb{N}$ as parameter. K picks n secret values $(x_1, \ldots, x_n) \in_R \mathbb{Z}^n_{ord(G)}$ computes corresponding contributions $z_i = g^{x_i}$ for all $i \in [1, n]$, performs n instances of the interactive algorithm $\mathcal{CGKA}.Setup(x_i, z_i)$ to obtain the tracing trapdoor $\hat{x}$ and auxiliary information aux_i for each instance, and sets $\mathcal{K} = \{z_i | i \in [1, n]\}$. K answers with the communication transcript $tscript$ of all protocol messages. Note that $tscript$ contains only public data sent over simulated communication channel and due to the security of $CGKA$ does not reveal any information that may lead to the computation of secrets x_i, $i \in [1, n]$ or $\hat{x}$. Among other information $tscript$ contains the set of public contributions $Z = \{z_i | i \in [1, n]\}$.

Join$_i$. This query can be used to introduce a new member to the group that is not controlled by K. K performs $|\mathcal{K}|$ instances of the interactive algorithm $\mathcal{CGKA}.Join_i(x_i, aux_i)$ and obtains the tracing trapdoor $\hat{x}$ and auxiliary information aux_i for each instance. K answers with the communication transcript $tscript$ of all protocol messages. Note that $tscript$ contains an updated set of public contributions $Z = \{z_i | i \in [1, n + 1]\}$ including the contribution of the joined member.

Leave. This query can be used to exclude a member j from the group. It contains $z_j \in G$ as parameter. If $z_j \in \mathcal{K}$ then $\mathcal{K} = \mathcal{K} - \{z_j\}$. K performs $|\mathcal{K}|$ instances of the interactive algorithm $\mathcal{CGKA}.Leave(x_i, z_j, aux_i)$, and obtains the tracing trapdoor $\hat{x}$ and auxiliary information aux_i for each instance. K answers with the communication transcript $tscript$ of all protocol messages. Note that $tscript$ contains an updated set of public contributions $Z = \{z_i | i \in [1, n - 1]\}$.

SGroupKey. This query can be used to obtain the tracing trapdoor $\hat{x}$ and contains no parameters. K answers with $\hat{x}$. Note that due to the security requirements of $CGKA$ the tracing trapdoor $\hat{x}$ does not reveal any information that can be used to compute any member's secret x_i.

SMemberKey. This query can be used to obtain a member's i secret x_i. It contains $z_i \in Z$ as parameter. K answers with x_i.

Interaction between B and A. Assume that there exists a polynomial-time adversary A that wins the traceability game in Section 2.3 with non-negligible probability, i.e., $\textbf{Adv}^{\text{tr}}_A > \epsilon$. Let $\mathcal{A}_{[t]}$ denote a set of contributions of group members controlled by the adversary, $\mathcal{A}'_{[t]}$ a set of contributions of group members corrupted by the adversary, and $\mathcal{B}_{[t]}$ a set of contributions of group members that are not controlled and not corrupted by the adversary in the group formation that is identified with the counter value t, and n be a number of group members of this group formation, such that the following relations hold for all t: $\mathcal{A}_{[t]} \cup \mathcal{B}_{[t]} = Z_{[t]}$, $\mathcal{A}_{[t]} \cap \mathcal{B}_{[t]} = \emptyset$, $\mathcal{A}'_{[t]} \subseteq \mathcal{B}_{[t]}$, and $n = |\mathcal{A}_{[t]}| + |\mathcal{B}_{[t]}|$. We stress that group members in $\mathcal{B}_{[t]}$ are controlled by the oracle K, and that all secret signing keys $x_{i[t]}$ of group members in $\mathcal{B}_{[t]}$ are initially known only to K and not to B.

Setup: B picks a random $n \in_R \mathbb{N}$, sets counter value $t = 0$, initializes sets $\mathcal{A}_{[0]} = \emptyset$, $\mathcal{A}'_{[0]} = \emptyset$, queries K on *Setup* with n and obtains communication transcript $tscript$, that reveals the set of contributions $Z_{[0]} = \{z_{i[0]} | i \in [1, n]\}$. B defines $\mathcal{B}_{[0]} = Z_{[0]}$, queries K on *SGroupKey* and obtains the tracing trapdoor $\hat{x}_{[0]}$. B computes $\hat{y}_{[0]} = g^{\hat{x}_{[0]}}$, and gives A the group public key $Y_{[0]} = (\hat{y}_{[0]}, Z_{[0]})$. (Note that B does not know any secret signing key $x_{i[0]}$ used to compute the corresponding contribution.)

Hash Queries: At any time A can query the hash function H. B answers the query completely at random while keeping constistency.

Queries:
Join. A picks $x_{a[t+1]} \in \mathbb{Z}_{ord(G)}$, computes contribution $z_{a[t+1]} = g^{x_{a[t+1]}}$, and starts the interactive algorithms $\mathcal{CGKA}.Join_u(x_{a[t+1]}, z_{a[t+1]})$ and $\mathcal{CGKA}.Join_i(x_{a_i[t]}, aux_{a_i})$ for all a_i with $z_{a_i[t]} \in \mathcal{A}_{[t]}$, whereas K upon B's query *Join*$_i$ starts $\mathcal{CGKA}.Join_i(x_{i[t]}, aux_{i[t]})$ for all i with $z_{i[t]} \in \mathcal{B}_{[t]}$. B forwards A's messages to K and vice versa until the protocol is finished. For the instance of $\mathcal{CGKA}.Join_u()$ A obtains $\hat{x}_{[t+1]}$ and $aux_{a[t+1]}$, and for all instances of $\mathcal{CGKA}.Join_i()$ it obtains $\hat{x}_{[t+1]}$, $aux_{a_i[t+1]}$ and possibly updated $x_{a_i[t+1]}$, whereas K obtains $\hat{x}_{[t+1]}$, $aux_{i[t+1]}$ and possibly updated $x_{i[t+1]}$. B receives $tscript$ von K that includes the set of updated contributions $Z_{[t+1]} = \{z_{i[t+1]} | i \in [1, n+1]\}$, and updates $\mathcal{A}_{[t+1]} = \mathcal{A}_{[t]} + \{z_{a[t+1]}\}$, $\mathcal{A}'_{[t+1]} = \mathcal{A}'_{[t]}$ and $\mathcal{B}_{[t+1]} = \mathcal{B}_{[t]}$ accordingly. B queries K on *SGroupKey* and obtains $\hat{x}_{[t+1]}$. Both, A and B compute $\hat{y}_{[t+1]} = g^{\hat{x}_{[t+1]}}$ and $Y_{[t+1]} = (\hat{y}_{[t+1]}, Z_{[t+1]})$.

Leave. Suppose member $j \in [1, n]$ should be excluded from the group. If $z_{j[t]} \in \mathcal{A}_{[t]}$ then B updates $\mathcal{A}_{[t+1]} = \mathcal{A}_{[t]} - \{z_{j[t]}\}$, else $\mathcal{A}_{[t+1]} = \mathcal{A}_{[t]}$. If $z_{j[t]} \in \mathcal{B}_{[t]}$ then $\mathcal{B}_{[t+1]} = \mathcal{B}_{[t]} - \{z_{j[t]}\}$, else $\mathcal{B}_{[t+1]} = \mathcal{B}_{[t]}$. If $z_{j[t]} \in \mathcal{A}'_{[t]}$ then $\mathcal{A}'_{[t+1]} = \mathcal{A}'_{[t]} - \{z_{j[t]}\}$, else $\mathcal{A}'_{[t+1]} = \mathcal{A}'_{[t]}$. A starts interactive algorithms $\mathcal{CGKA}.Leave(x_{a_i[t]}, z_{j[t]}, aux_{a_i})$ for all a_i with $z_{a_i[t]} \in \mathcal{A}_{[t+1]}$, whereas K upon B's query *Leave* with parameter $z_{j[t]}$ starts $\mathcal{CGKA}.Leave(x_{i[t]}, z_{j[t]}, aux_i)$ for all members i with $z_{i[t]} \in \mathcal{B}_{[t+1]}$. B forwards A's messages to K and vice versa until the protocol is finished. A obtains $\hat{x}_{[t+1]}$, $aux_{a_i[t+1]}$ and possibly updated $x_{a_i[t+1]}$, whereas K obtains $\hat{x}_{[t+1]}$, $aux_{i[t+1]}$ and possibly updated $x_{i[t+1]}$. B receives $tscript$ von K that includes the set of updated contributions $Z_{[t+1]} = \{z_{i[t+1]} | i \in [1, n-1]\}$, and updates possibly changed contributions in $\mathcal{A}_{[t+1]}$ and $\mathcal{B}_{[t+1]}$ using $Z_{[t+1]}$. B queries K on *SGroupKey* and obtains $\hat{x}_{[t+1]}$. Both, A and B compute $\hat{y}_{[t+1]} = g^{\hat{x}_{[t+1]}}$ and $Y_{[t+1]} = (\hat{y}_{[t+1]}, Z_{[t+1]})$.

CorruptMember. If A queries B with a contribution $z_{i[t]} \in \mathcal{B}_{[t]}$ then B forwards it to K as *SMemberKey* query and obtains the corresponding secret signing key $x_{i[t]}$ that it returns to A. B updates $\mathcal{A}'_{[t]} = \mathcal{A}'_{[t]} + \{z_{i[t]}\}$. However, if $z_{i[t]} \in \mathcal{A}_{[t]}$ then interaction outputs a failure. Note that B learns all secret signing keys that correspond to contributions in $\mathcal{A}'_{[t]}$.

CorruptGroupKey. B answers with $\hat{x}_{[t]}$ that it has previously obtained from K.

Sign. B is given a message $m \in \{0,1\}^*$, a contribution $z_{j[t]}$ and a counter value t. If $z_{j[t]} \in \mathcal{A}_{[t]}$ then interaction outputs a failure. If $z_{j[t]} \in \mathcal{A}'_{[t]}$ then B computes the signature $\sigma = Sign(x_{j[t]}, m, Y_{[t]})$ and gives it to A (note, B knows $x_{j[t]}$ for all contributions in $\mathcal{A}'_{[t]}$). If $z_{j[t]} \in \mathcal{B}_{[t]} - \mathcal{A}'_{[t]}$ then B generates the signature σ without knowing $x_{j[t]}$ as follows. B picks a random $r \in_R \mathbb{Z}_{ord(G)}$, computes $\tilde{g} = g^r$, and $\tilde{y} = \hat{y}_{[t]}^r z_{j[t]}$. B computes the signature of knowledge $S = SK[(\alpha_i, \beta) : \tilde{g} = g^\beta \wedge \tilde{y} = \hat{y}_{[t]}^\beta g^{\alpha_i} \wedge (z_{1[t]} = g^{\alpha_1} \vee \ldots \vee z_{n[t]} = g^{\alpha_n})](m)$ without knowing the corresponding exponent α_i using the random oracle simulation as follows:
∗ $(s_{u_1}, \ldots, s_{u_n}, s_{v_1}, \ldots, s_{v_n}) \in_R \mathbb{Z}^{2n}_{ord(G)}$; $(c_1, \ldots, c_n) \in_R (\{0,1\}^k)^n$; $\bar{c} = \bigoplus_{i=1}^n c_i$;
∗ for all $i \in [1, n]$: $u_i = \tilde{y}^{c_i} \hat{y}_{[t]}^{s_{u_i}} g^{s_{v_i}}$, $v_i = z_{i[t]}^{c_i} g^{s_{v_i}}$, $w_i = \tilde{g}^{c_i} g^{s_{u_i}}$;
∗ set $H(g, \hat{y}_{[t]}, \tilde{g}, \tilde{y}, u_1, \ldots, u_n, v_1, \ldots, v_n, w_1, \ldots, w_n, m) := \bar{c}$ if the hash oracle has not yet been queried on these values and $\bar{c}$ has not been previously returned; otherwise B reselects the randoms;
∗ $S = (c_1, \ldots, c_n, s_{u_1}, \ldots, s_{u_n}, s_{v_1}, \ldots, s_{v_n})$;
B gives the generated signature $\sigma = (\tilde{g}, \tilde{y}, S)$ to A.

Output: Finally, A outputs a successful forgery consisting of a message m, signature $\sigma = (\tilde{g}, \tilde{y}, S)$, and a counter value t. The interaction outputs (m, σ, t).

Forgery Types. We distinguish between the following two successful forgery types with respect to the output of the traceability game in Section 2.3.

Type 1. The forgery (m, σ, t) is accepted by the verification algorithm, i.e., $Verify(\sigma, m, Y_{[t]}) = 1$, but causes the tracing algorithm $Trace(\sigma, m, Y_{[t]}, \hat{x}_{[t]})$ to fail. A closer look on the tracing algorithm reveals that such failure can only occur if for $z_{i[t]} = \tilde{y}/V_1$ with $V_1 = \tilde{g}^{\hat{x}[t]}$ the relation $z_{i[t]} \notin Z_{[t]}$ holds. This implies that either $(\tilde{g}, \tilde{y})$ is not a correct encryption of a contribution or that the encrypted contribution is not part of $Z_{[t]}$, thus S has to be a forged signature of knowledge; otherwise the signature σ would have been rejected by the verification algorithm. Assuming the security of applied signatures of knowledge A outputs the forgery of type 1 only with some negligible probability ϵ_1.

Type 2. On input the forgery (m, σ, t) the tracing algorithm $Trace(\sigma, m, Y_{[t]}, \hat{x}_{[t]})$ returns the encoded contribution $z_{i[t]} \in \mathcal{B}_{[t]} - \mathcal{A}'_{[t]}$. In the following we show that if A returns a forgery of type 2 then B is able to compute the discrete logarithm of the encoded contribution $z_{i[t]}$ to the base g in the group $G = <g>$. We modify the interaction between B and A and show that the forking lemma for adaptively chosen-message attacks from [19, Theorem 3] can be applied.

The forking lemma can be applied to the signatures of the form (σ_1, c, σ_2) on a message m where σ_1 depends only on the random chosen values, c is a hash value that depends on m and σ_1, and σ_2 depends only on σ_1, m, and c. We abbreviate the signature $\sigma = (\tilde{g}, \tilde{y}, S)$ of a message m produced by the signing algorithm of the group signature scheme in Section 3.3 as $\sigma = (\sigma_1, c, \sigma_2)$ with the following parameters:

- $\sigma_1 = (\tilde{g}, \tilde{y}, u_1, \ldots, u_n, v_1, \ldots, v_n, w_1, \ldots, w_n)$
- $c = c_1 \oplus \ldots \oplus c_n$ with $c = H(g, \hat{y}_{[t]}, \sigma_1, m)$
- $\sigma_2 = (s_{u_1}, \ldots, s_{u_n}, s_{v_1}, \ldots, s_{v_n})$

The forking lemma for adaptively chosen-message attacks requires that the signature $\sigma = (\sigma_1, c, \sigma_2)$ can be simulated without knowing the corresponding secret signing key with an indistinguishable distribution probability. We stress that the computation of S described in the signing query of the interaction between B and A provides such simulation in the random oracle model.

Assume that interaction between B and A outputs a successful forgery (m, σ, t) of type 2, where $\sigma = (\sigma_1, c, \sigma_2)$. The probability that the interaction outputs the forgery of type 2 with A having queried the hash oracle on $(m, g, \hat{y}_{[t]}, \sigma_1)$ is $\epsilon_2 = (\mathbf{Adv}_A^{\text{tr}} - \epsilon_1)(1 - 2^{-k})$, where 2^{-k} stands for the possibility that A guessed the corresponding value $c = \bigoplus_{j=1}^{n} c_j$ without querying H. Let $z_{i[t]} \in Z_{[t]}$ be a contribution returned by the tracing algorithm on the forgery (m, σ, t).

By the forking lemma we can rewind the interaction and A to the moment of this query, repeat the interaction with different responses of H, and obtain a second successful forgery (m, σ', t) where $\sigma' = (\sigma_1, c', \sigma'_2)$ so that $c' \neq c$ and $\sigma'_2 \neq \sigma_2$ with the probability at least ϵ_2. Note that $c' = \bigoplus_{j=1}^{n} c'_j$ such that $c'_i \neq c_i$, whereas values c_j and c'_j with $j \neq i$ are independent from the response of the hash oracle as shown in the signing algorithm of the group signature scheme, and are not used in the computation of the discrete logarithm. We compute the corresponding secret signing key $x_{i[t]} = \log_g z_{i[t]}$ from the equality $z_{i[t]}^{c_i} g^{s_{v_i}} = z_{i[t]}^{c'_i} g^{s'_{v_i}}$ as $x_{i[t]} = \frac{s'_{v_i} - s_{v_i}}{c_i - c'_i}$. Hence, using A we can compute the discrete logarithm in G with a non-negligible probability $((\mathbf{Adv}_A^{\text{tr}} - \epsilon_1)(1 - 2^{-k}))^2$. This is a contradiction to the Discrete Logarithm assumption. Thus, $\mathbf{Adv}_A^{\text{tr}} \leq \epsilon$ and $\mathcal{DGS}$ is traceable according to Definition 3. □

Remark 5. Most of the CGKA protocols do not achieve Perfect Forward Secrecy (PFS) since not all contributions $z_{i[t]}$ are changed every time a group formation changes, and, therefore, the exposure of one $x_{i[t]}$ can possibly reveal tracing trapdoors of multiple sessions until $x_{i[t]}$ is changed. Note that although we allow the *CorruptMember* query, the absence of PFS is not a hazard for the traceability, since the knowledge of the group key (allowed by *CorruptGroupKey* query) is not sufficient to break the requirement as shown in the proof.

A.3 Proof of Lemma 3 (Anonymity)

Suppose there exists a polynomial-time adversary A that breaks the anonymity of $\mathcal{GS}$. In the following we show that it is possible to construct a polynomial-time distinguisher D against the security of $CGKA$

(decisional group key secrecy requirement), and adversary B against the DDH assumption in G such that

$$\mathbf{Adv}_A^{\text{an}} \leq 2\mathbf{Adv}_D^{\text{cgka}} + 2\mathbf{Adv}_B^{\text{ddh}} + 2\epsilon,$$

where $\mathbf{Adv}_D^{\text{cgka}} = Pr[D(tscript, \chi_{1[t]}) = 1] - Pr[D(tscript, \chi_{0[t]}) = 1]$ is the advantage of D in distinguishing a secret group key (i.e., $\chi_{1[t]} = \hat{x}_{[t]}$) computed by $CGKA$ from a random number (i.e., $\chi_{0[t]} \in_R \mathbb{Z}_{ord(G)}$). Recall that $tscript$ is the transcript of all protocol messages that contains also the set of public contributions $Z_{[t]} = \{z_{i[t]} | i \in [1, n]\}$. Since we assume that $CGKA$ is a secure protocol suite, and the DDH assumption holds in G the right hand side of the inequality is negligible so, the advantage on the left side is also negligible, and $\mathcal{DGS}$ is anonymous according to Definition 4.

Construction of D. Oracle K introduced in the proof of traceability can be used to generate inputs to D and perform $CGKA$ protocols on D's queries. However, we modify K to suit the anonymity requirement as follows. K does not answer to *SGroupKey* and *SMemberKey* queries, otherwise D would be able to distinguish the tracing trapdoor simply using these queries. Let $\mathcal{K}_{[t]}$ and $\mathcal{A}_{[t]}$ denote sets of contributions of members controlled by K and A during the group formation identified by t, respectively. K proceeds queries *Setup*, *Join*$_i$ and *Leave* as described in A.2. Additionally, at the end of each query if $\mathcal{A}_{[t]} = \emptyset$ then K chooses a random bit $e \in \{0, 1\}$ and together with $tscript$ returns $\chi_{e[t]}$ to D, such that $\chi_{e[t]}$ is either the secret group key $\hat{x}_{[t]}$ if $e = 1$ or a random value $\hat{r}_{[t]} \in_R \mathbb{Z}_{ord(G)}$ if $e = 0$. However, if $\mathcal{A}_{[t]} \neq \emptyset$ then K returns $tscript$ and $\hat{x}_{[t]}$ (this is because A knows $\hat{x}_{[t]}$ too). D wins the game if for at least one $\chi_{e[t]}$ it can correctly guess e. $CGKA$ is said to fulfill decisional group key secrecy requirement if $\mathbf{Adv}_D^{\text{cgka}} = Pr[D(tscript, \chi_{1[t]}) = 1] - Pr[D(tscript, \chi_{0[t]}) = 1]$ is negligible. In the following game we show, how D uses A to break the security of $CGKA$. For clarity we omit the operations performed by K and describe the views of D and A.

Setup: D sets counter value $t = 0$, queries K on *Setup* and obtains $Z_{[0]} = \{z_{i[0]} | i \in [1, n]\}$ and $\chi_{e[0]}$. It computes $\hat{y}_{[0]} = g^{\chi_{e[0]}}$, initializes set $\mathcal{A}_{[0]} = \emptyset$, and gives A the group public key $Y_{[0]} = (\hat{y}_{[0]}, Z_{[0]})$. (Note that $\hat{y}_{[0]}$ is either $g^{\hat{x}_{[0]}}$ or $g^{\hat{r}_{[0]}}$ depending on e.)

Hash Queries: At any time A can query the hash function H. B answers the query completely at random while keeping constistency.

Type1-Queries:
Join. A picks $x_{a[t+1]} \in \mathbb{Z}_{ord(G)}$, computes $z_{a[t+1]} = g^{x_{a[t+1]}}$, and starts $\mathcal{CGKA}.Join_u(x_{a[t+1]}, z_{a[t+1]})$ and $\mathcal{CGKA}.Join_i(x_{a_i[t]}, z_{a_i[t]}, aux_{a_i[t]})$ for all a_i with $z_{a_i[t]} \in \mathcal{A}_{[t]}$. D forwards A's messages to K and vice versa until the protocol is finished. As result of $\mathcal{CGKA}.Join_u()$ A obtains $\hat{x}_{[t+1]}$ and $aux_{a[t+1]}$, and for all $\mathcal{CGKA}.Join_i()$ it obtains $\hat{x}_{[t+1]}$, $aux_{a_i[t+1]}$ and possibly updated $x_{a_i[t+1]}$, whereas D obtains $Z_{[t+1]}$ and $\hat{x}_{[t+1]}$ from K. A and D compute $\hat{y}_{[t+1]} = g^{\hat{x}_{[t+1]}}$ and $Y_{[t+1]} = (\hat{y}_{[t+1]}, Z_{[t+1]})$. D updates $\mathcal{A}_{[t+1]} = \mathcal{A}_{[t]} + \{z_{a[t+1]}\}$.

Leave. Suppose member $j \in [1, n]$ should be excluded from the group. If $z_{j[t]} \in \mathcal{A}_{[t]}$ then D updates $\mathcal{A}_{[t+1]} = \mathcal{A}_{[t]} - \{z_{j[t]}\}$, else it sets $\mathcal{A}_{[t+1]} = \mathcal{A}_{[t]}$.

- Case $\mathcal{A}_{[t+1]} \neq \emptyset$: A starts $\mathcal{CGKA}.Leave(x_{a_i[t]}, z_{j[t]}, aux_{a_i[t]})$ for all a_i with $z_{a_i[t]} \in \mathcal{A}_{[t+1]}$. D forwards A's messages to K and vice versa until the protocol is finished. A obtains $Y_{[t+1]}$, $\hat{x}_{[t+1]}$ and all secret signing keys $x_{a_i[t+1]}$, whereas D obtains $Z_{[t+1]}$ and $\hat{x}_{[t+1]}$ from K. A and D compute $\hat{y}_{[t+1]} = g^{\hat{x}_{[t+1]}}$ and $Y_{[t+1]} = (\hat{y}_{[t+1]}, Z_{[t+1]})$. D updates possibly changed contributions in $\mathcal{A}_{[t+1]}$.
- Case $\mathcal{A}_{[t+1]} = \emptyset$: D queries K on *Leave* with parameter $z_{j[t]}$, and obtains $Z_{[t+1]}$ and $\chi_{e[t+1]}$. It computes $\hat{y}_{[t+1]} = g^{\chi_{e[t+1]}}$, and gives A the group public key $Y_{[t+1]} = (\hat{y}_{[t+1]}, Z_{[t+1]})$. (Note that $\hat{y}_{[t+1]}$ is either $g^{\hat{x}_{[t+1]}}$ or $g^{\hat{r}_{[t+1]}}$ depending on e.)

Sign. D is given a message $m \in \{0, 1\}^*$, a member $j \in [1, n]$ and a counter value t. If $z_{j[t]} \in \mathcal{A}_{[t]}$ then B aborts. Else D generates a signature for m using j's contribution as follows. B picks a random $r \in_R \mathbb{Z}_{ord(G)}$, computes $\tilde{g} = g^r$ and $\tilde{y} = \hat{y}_{[t]}^r z_{j[t]}$. B computes the signature of knowledge $S = SK[(\alpha_i, \beta) : \tilde{g} = g^\beta \wedge \tilde{y} = \hat{y}_{[t]}^\beta g^{\alpha_i} \wedge (z_{1[t]} = g^{\alpha_1} \vee \ldots \vee z_{n[t]} = g^{\alpha_n})](m)$ without knowing the corresponding exponent α_i using the random oracle simulation as described in the query ***Sign*** of the traceability game in A.2, and gives the generated signature $\sigma = (\tilde{g}, \tilde{y}, S)$ to A.

Challenge: D is given a message $m' \in \{0,1\}^*$, two members i_0 and i_1, and a counter value t. If $\mathcal{A}_{[t]} \neq \emptyset$ then D aborts. Otherwise, D picks a random bit $d \in_R \{0,1\}$, generates the signature $\sigma_d = (\tilde{g}, \tilde{y}, S)$ using contribution $z_{i_d[t]}$ as described in the query ***Sign*** of Type1, and gives it to A. (Note that since D responds to the challenge only if $\mathcal{A}_{[t]} = \emptyset$, $\chi_{e[t]}$ is either $\hat{x}_{[t]}$ or $\hat{r}_{[t]}$ depending on e.)

Type2-Queries: D responds to the possible queries of A as described in Type1-Queries.

Output: Eventually, A outputs a bit $d' \in \{0,1\}$. If $d' = d$ then D outputs 1 (indicating that $\chi_{e[t]} = \hat{x}_{[t]}$); otherwise it outputs 0 (indicating that $\chi_{e[t]} = \hat{r}_{[t]}$).

Case $e = 1$. The most important observation in this case is that $\chi_{1[t]}$ is a tracing trapdoor $\hat{x}_{[t]}$ computed in the sense of $CGKA$. Hence, $Y_{[t]} = (\hat{y}_{[t]}, Z_{[t]})$ is equivalent to the group public key of the proposed scheme in Section 3.3. Therefore, signatures generated by D in the random oracle model are indistinguishable from those in the anonymity game in Section 2.3. Obviously, D outputs 1 whenether A correctly guesses bit d. Hence,

$$\begin{aligned} Pr[D(tscript, \chi_{1[t]}) = 1] &= Pr[A(\sigma_1) = 1]Pr[d = 1] + Pr[A(\sigma_0) = 0]Pr[d = 0] \\ &= \frac{1}{2}(Pr[A(\sigma_1) = 1] + Pr[A(\sigma_0) = 0]) \\ &= \frac{1}{2}(Pr[A(\sigma_1) = 1] + 1 - Pr[A(\sigma_0) = 1]) \\ &= \frac{1}{2}(1 + \mathbf{Adv}_A^{\text{an}}) \\ &= \frac{1}{2} + \frac{1}{2}\mathbf{Adv}_A^{\text{an}} \end{aligned} \quad (1)$$

Case $e = 0$. The most important observation in this case is that $\chi_{0[t]}$ is a random value $\hat{r}_{[t]} \in_R \mathbb{Z}_{ord(G)}$. Note that in both cases $\hat{y}_{[t]}$ is constructed by D as $g^{\chi_{e[t]}}$. Hence in this case, $Y_{[t]} = (\hat{y}_{[t]}, Z_{[t]})$ is a simulated group public key. Therefore, signature σ_d generated by D is a signature-like looking tuple. These signatures can be classified into two distributions E_d, $d \in \{0,1\}$ depending on the choice of d. Obviously, D outputs 1 whenether A can distinguish whether σ_d belongs to distribution E_0 or E_1. Hence, $Pr[D(tscript, \chi_{0[t]}) = 1] = Pr[A(E_1) = 1]Pr[d = 1] + Pr[A(E_0) = 0]Pr[d = 0]$, i.e.,

$$Pr[D(tscript, \chi_{0[t]}) = 1] = \frac{1}{2}(Pr[A(E_1) = 1] + Pr[A(E_0) = 0]) \quad (2)$$

Instead of precise estimation of this probability we relate it to the probability of breaking the DDH assumption by the adversary B, assuming that there exists adversary A that can distinguish between signatures sampled from distributions E_0 and E_1.

Construction of B. Adversary B is given a tuple $(g, T_a = g^a, T_b = g^b, T_c) \in G^4$ where $a, b \in_R \mathbb{Z}_{ord(G)}$ and either $T_c = g^c$ with $c \in_R \mathbb{Z}_{ord(G)}$, or $T_c = g^{ab}$. B decides which T_c it was given by interacting with A as follows. Let $\mathcal{A}_{[t]}$ denote a set of contributions of group members controlled by the adversary, and $\mathcal{B}_{[t]}$ a set of contributions of group members that are not controlled by A in the group formation that is identified by the counter value t, respectively, and n be a number of group members of this group formation. Note that $\mathcal{A}_{[t]} \cup \mathcal{B}_{[t]} = Z_{[t]}$, $\mathcal{A}_{[t]} \cap \mathcal{B}_{[t]} = \emptyset$, and $n = |\mathcal{A}_{[t]}| + |\mathcal{B}_{[t]}|$.

Setup: B picks a random $n \in_R \mathbb{N}$, sets counter value $t = 0$, picks randoms $(\hat{u}_{[0]}, x_{1[0]}, \ldots, x_{n[0]}) \in_R \mathbb{Z}_{ord(G)}^{n+1}$. It computes $\hat{y}_{[0]} = T_a^{\hat{u}_{[0]}}$ and $Z_{[0]} = \{g^{x_{i[0]}} | \forall i \in [1, n]\}$, performs n instances of the interactive algorithm $\mathcal{CGKA}.Setup(x_{i[0]}, z_{i[0]})$ in parallel, and obtains $\hat{x}_{[0]}$ and $aux_{i[0]}$ for each instance. It initializes set $\mathcal{A}_{[0]} = \emptyset$, set $\mathcal{B}_{[0]} = Z_{[0]}$, and gives A the group public key $Y_{[0]} = (\hat{y}_{[0]}, Z_{[0]})$. (Note that $\hat{y}_{[0]}$ is computed as $g^{a\hat{u}_{[0]}}$ and not using the obtained secret key $\hat{x}_{[0]}$, and that B cannot compute $a\hat{u}_{[0]}$ because it does not know a. Obviously, $Y_{[0]}$ corresponds to the simulated group public key from the construction of the distinguisher D above, allowing us later to relate the probabilities of both games.)

Hash Queries: At any time A can query the hash function H. B answers the query completely at random while keeping constistency.

Type1-Queries:
Join. A picks $x_{a[t+1]} \in \mathbb{Z}_{ord(G)}$, computes contribution $z_{a[t+1]} = g^{x_{a[t+1]}}$, and starts instances of the interactive algorithms $\mathcal{CGKA}.Join_u(x_{a[t+1]}, z_{a[t+1]})$ and $\mathcal{CGKA}.Join_i(x_{a_i[t]}, z_{a_i[t]}, aux_{a_i[t]})$ for all a_i with $z_{a_i[t]} \in \mathcal{A}_{[t]}$, whereas B starts $\mathcal{CGKA}.Join_i(x_{i[t]}, z_{i[t]}, aux_{i[t]})$ for all i with $z_{i[t]} \in \mathcal{B}_{[t]}$. After the protocol is completed A obtains $\hat{x}_{[t+1]}$, $aux_{a[t+1]}$, and possibly updated $x_{a_i[t]}$, whereas B obtains $\hat{x}_{[t+1]}$, $aux_{i[t+1]}$ and possibly updated $x_{i[t+1]}$. Both, A and B compute $\hat{y}_{[t+1]} = g^{\hat{x}_{[t+1]}}$ and $Y_{[t+1]} = (\hat{y}_{[t+1]}, Z_{[t+1]})$. B adds the contribution of the introduced member to the set $\mathcal{A}_{[t]}$, i.e., $\mathcal{A}_{[t+1]} = \mathcal{A}_{[t]} + \{z_{a[t+1]}\}$.
Leave. Suppose member $j \in [1, n]$ should be excluded from the group. If $z_{j[t]} \in \mathcal{A}_{[t]}$ then B updates $\mathcal{A}_{[t+1]} = \mathcal{A}_{[t]} - \{z_{j[t]}\}$ and $\mathcal{B}_{[t+1]} = \mathcal{B}_{[t]}$, else it sets $\mathcal{A}_{[t+1]} = \mathcal{A}_{[t]}$ and $\mathcal{B}_{[t+1]} = \mathcal{B}_{[t]} - \{z_{j[t]}\}$.

- Case $\mathcal{A}_{[t+1]} \neq \emptyset$: A starts $\mathcal{CGKA}.Leave(x_{a_i[t]}, z_{j[t]}, aux_{a_i[t]})$ for all a_i with $z_{a_i[t]} \in \mathcal{A}_{[t+1]}$, whereas B starts $\mathcal{CGKA}.Leave(x_{i[t]}, z_{j[t]}, aux_{i[t]})$ for all i with $z_{i[t]} \in \mathcal{B}_{[t+1]}$. After the protocol is completed A obtains $Y_{[t+1]}$, $\hat{x}_{[t+1]}$ and all secret signing keys $x_{a_i[t+1]}$, whereas B obtains $Y_{[t+1]}$, $\hat{x}_{[t+1]}$ and all secret signing keys $x_{i[t+1]}$. B updates possibly changed contributions in $\mathcal{A}_{[t+1]}$ and $\mathcal{B}_{[t+1]}$.
- Case $\mathcal{A}_{[t+1]} = \emptyset$: B starts $\mathcal{CGKA}.Leave(x_{i[t]}, z_{j[t]}, aux_{i[t]})$ for all i with $z_{i[t]} \in \mathcal{B}_{[t+1]}$, and obtains $\hat{x}_{[t+1]}$, $aux_{i[t+1]}$ and possibly changed $x_{i[t+1]}$. It picks a random $\hat{u}_{[t+1]} \in_R \mathbb{Z}_{ord(G)}$, computes $\hat{y}_{[t+1]} = T_a^{\hat{u}_{[t+1]}}$, and gives A the group public key $Y_{[t+1]} = (\hat{y}_{[t+1]}, Z_{[t+1]})$. B updates possibly changed contributions in $\mathcal{B}_{[t+1]}$. (Note that like in setup $Y_{[t+1]}$ is a simulated group public key.)

Sign. B is given a message $m \in \{0,1\}^*$, a member $j \in [1, n]$ and a counter value t. If $\mathcal{A}_{[t]} \neq \emptyset$ and $z_{j[t]} \notin \mathcal{A}_{[t]}$ then B computes the signature $\sigma = Sign(x_{j[t]}, m, Y_{[t]})$ and gives it to A. If $\mathcal{A}_{[t]} \neq \emptyset$ and $z_{j[t]} \in \mathcal{A}_{[t]}$ then B aborts. Else if $\mathcal{A}_{[t]} = \emptyset$ then B generates a signature for m using j's contribution as follows. B picks a random $r \in_R \mathbb{Z}_{ord(G)}$, computes $\tilde{g} = T_b^r$ (note that $\tilde{g} = g^{br}$), and $\tilde{y} = T_c^{\hat{u}_{[t]}r} z_{j[t]}$ (note that if $T_c = g^{ab}$ then $\tilde{y} = g^{ab\hat{u}_{[t]}r} z_{j[t]} = \hat{y}_{[t]}^{br} z_{j[t]}$; otherwise if $T_c = g^c$ then $\tilde{y} = \hat{y}_{[t]}^{c^*} z_{j[t]}$ for some unkonwn random $c^* = cr/a$). B computes the signature of knowledge $S = SK[(\alpha_i, \beta) : \tilde{g} = g^\beta \wedge \tilde{y} = \hat{y}_{[t]}^\beta g^{\alpha_i} \wedge (z_{1[t]} = g^{\alpha_1} \vee \ldots \vee z_{n[t]} = g^{\alpha_n})](m)$ without knowing the corresponding exponent β using the same random oracle simulation as described in query ***Sign*** in the traceability game in A.2, and gives the generated signature $\sigma = (\tilde{g}, \tilde{y}, S)$ to A.

Challenge: B is given a message $m' \in \{0,1\}^*$, two members i_0 and i_1, and a counter value t. If $\mathcal{A}_{[t]} \neq \emptyset$ then B aborts. Otherwise, B picks a random bit $d \in_R \{0,1\}$, generates the signature $\sigma_d = (\tilde{g}, \tilde{y}, S)$ using contribution $z_{i_d[t]}$ as described in the query *Sign* of Type1, and gives it to A.

Type2-Queries: B responds to the possible queries of A as described in Type1-Queries.

Output: Eventually, A outputs a bit $d' \in \{0,1\}$. If $d' = d$ then B outputs 1 (indicating that $T_c = g^{ab}$); otherwise it outputs 0 (indicating that $T_c = g^c$).

Recall that the advantage of B in breaking the DDH assumption is defined in Section 3.1 as $\mathbf{Adv}_B^{\mathsf{ddh}} = Pr[B(g, T_a, T_b, g^{ab}) = 1] - Pr[B(g, T_a, T_b, g^c) = 1]$. In the following we compute both probabilities and relate them to the probabilities obtained from the game with distinguisher D in the previous paragraph.

Case $T_c = g^{ab}$. The most important observation in this case is that the signature σ_d generated by B in the random oracle model is computed based on the simulated group public key $Y_{[t]} = (\hat{y}_{[t]}, Z_{[t]})$, i.e., $\tilde{g} = g^{br}$ and $\tilde{y} = \hat{y}_{[t]}^{br} z_{i_d[t]}$. Therefore, signature σ_d is sampled from the distribution E_d introduced in the previous paragraph. Obviously, B outputs 1 whenether A correctly guesses the distribution. Hence, $Pr[B(g, T_a, T_b, g^{ab}) = 1] = Pr[A(E_1) = 1]Pr[b = 1] + Pr[A(E_0) = 0]Pr[b = 0]$. Considering the Equation 2, we obtain

$$Pr[B(g, T_a, T_b, g^{ab}) = 1] = Pr[D(tscript, \chi_{0[t]}) = 1] \tag{3}$$

Case $T_c = g^c$. In this case the encryption of the contribution $z_{i_d[t]}$ in the signature σ_d given to A is not correct, i.e., $\tilde{g} = g^{br}$ and $\tilde{y} = \hat{y}_{[t]}^{c^*} z_{i_d[t]}$ for some unkown random $c^* = cr/a$. Obviously, value $\tilde{y}$ is indistinguishable from a random number. Thus, the probability of A in guessing bit d correctly is not greater than that of a random guess, i.e., $1/2 + \epsilon$. Hence,

$$Pr[B(g, T_a, T_b, g^c) = 1] \leq 1/2 + \epsilon \tag{4}$$

In the following we combine results from both parts of the proof. Recall that $\mathbf{Adv}_D^{\text{cgka}} = Pr[D(tscript, \chi_{1[t]}) = 1] - Pr[D(tscript, \chi_{0[t]}) = 1]$. With Equations 1 and 3 we obtain

$$\mathbf{Adv}_D^{\text{cgka}} = \frac{1}{2} + \frac{1}{2}\mathbf{Adv}_A^{\text{an}} - Pr[B(g, T_a, T_b, g^{ab}) = 1]$$

Addition of $Pr[B(g, T_a, T_b, g^c) = 1]$ on both sides of the above equation combined with $\mathbf{Adv}_B^{\text{ddh}} = Pr[B(g, T_a, T_b, g^{ab}) = 1] - Pr[B(g, T_a, T_b, g^c) = 1]$ implies

$$\mathbf{Adv}_D^{\text{cgka}} + Pr[B(g, T_a, T_b, g^c) = 1] = \frac{1}{2} + \frac{1}{2}\mathbf{Adv}_A^{\text{an}} - \mathbf{Adv}_B^{\text{ddh}}$$

By transformation and consideration of the Equation 4 we obtain the required inequality:

$$\begin{aligned}\mathbf{Adv}_A^{\text{an}} &= 2\mathbf{Adv}_D^{\text{cgka}} + 2\mathbf{Adv}_B^{\text{ddh}} + 2Pr[B(g, T_a, T_b, g^c) = 1] - 1 \\ &\leq 2\mathbf{Adv}_D^{\text{cgka}} + 2\mathbf{Adv}_B^{\text{ddh}} + 2\epsilon\end{aligned}$$

□