J Cryptol (2020) 33:1822–1870
https://doi.org/10.1007/s00145-020-09357-w

Journal of
CRYPTOLOGY

Foundations of Fully Dynamic Group Signatures*

Jonathan Bootle
University of California, Berkeley, Berkeley, CA, USA
jonathan.bootle@berkeley.edu

Andrea Cerulli
DFINITY, Zurich, Switzerland
andrea@dfinity.org

Pyrros Chaidos
National and Kapodistrian University of Athens, Athens, Greece
pchaidos@di.uoa.gr

Essam Ghadafi
University of the West of England, Bristol, UK
essam.ghadafi@uwe.ac.uk

Jens Groth
University College London, London, UK
DFINITY, Zurich, Switzerland
jens@dfinity.org

Communicated by Masayuki Abe

Received 23 October 2018 / Revised 13 February 2020
Online publication 2 June 2020

Abstract. Group signatures allow members of a group to anonymously sign on behalf of the group. Membership is administered by a designated group manager. The group manager can also reveal the identity of a signer if and when needed to enforce accountability and deter abuse. For group signatures to be applicable in practice, they need to support fully dynamic groups, i.e., users may join and leave at any time. Existing security definitions for fully dynamic group signatures are informal, have shortcomings, and are mutually incompatible. We fill the gap by providing a formal rigorous security model for fully dynamic group signatures. Our model is general and is not tailored toward a specific design paradigm and can therefore, as we show, be used to argue about the security of different existing constructions following different design paradigms. Our

*An extended abstract of this paper appeared in the Proceedings of Applied Cryptography and Network Security—ACNS 2016.

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007–2013)/ERC Grant Agreement No. 307937 and EPSRC Grant EP/J009520/1.

P. Chaidos: Was supported by an EPSRC scholarship (EP/G037264/1—Security Science DTC).

J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi: Most of the work was done while at University College London.

definitions are stringent and when possible incorporate protection against maliciously chosen keys. We consider both the case where the group management and tracing signatures are administered by the same authority, i.e., a single group manager, and also the case where those roles are administered by two separate authorities, i.e., a group manager and an opening authority. We also show that a specialization of our model captures existing models for static and partially dynamic schemes. In the process, we identify a subtle gap in the security achieved by group signatures using revocation lists. We show that in such schemes new members achieve a slightly weaker notion of traceability. The flexibility of our security model allows to capture such relaxation of traceability.

Keywords. Group signatures, Security definitions.

1. Introduction

Group signatures, put forward by Chaum and van Heyst [27], are a fundamental cryptographic primitive allowing a member of a group to anonymously sign messages on behalf of the group. Group membership is administered by a designated group manager. In the case of a dispute, the group manager, or a designated opening authority, has the ability to revoke the anonymity of a signature by revealing its signer.

In static group signatures [14], the group population is fixed once and for all during the setup phase. Partially dynamic group signatures [19,43] allow the enrollment of members in the group at any time, but members cannot leave or be removed from the group once they have joined. In many settings, however, it is desirable to offer full flexibility in joining and leaving the group, i.e., requiring fully dynamic group signatures. In this work, we address the fundamental question of defining security for fully dynamic group signatures.

1.1. *Background and Related Work*

Group Signatures Without Revocation. After their introduction, a very prolific line of research has emerged on group signatures. The first efficient construction of group signatures was given by Ateniese et al. [2] based on both the strong RSA assumption and the DDH assumption in the random oracle model [15]. At that time, however, the security of group signatures was not very well understood and early constructions were proved secure via informal arguments using various interpretations of their requirements.

To rectify this situation, Bellare et al. [14] formalized the security definitions for static groups where the group population is fixed once and for all during the setup phase. In their model, the group manager is also granted the authority of opening signatures and she is assumed to be fully trusted, except she may leak information to the adversary. Later on, Bellare et al. [19] and Kiayias and Yung [43] independently provided formal security definitions for partially dynamic groups, in which new group members can join the group at any time but cannot leave. The former model, following the suggestion of [26], separates the opening authority from the group manager, while the latter considers both roles overseen by the group manager. Aside from this, the main difference between the two models is that in [43] the group manager is trusted to give correct openings without providing proofs of attributions. More recently, Sakai et al. [59] extended the security

model of partially dynamic groups by including the notion of *opening soundness*, which ensures that a valid signature only traces to one user.

Ghadafi [35] gave a model for partially dynamic group signatures supporting threshold/distributed traceability where the role of the opening authority is distributed among n parties. Sakai et al. [31] extended the static group model by adding a dedicated authority, the admitter, who decides which subset of the message space signatures on which could be opened by the opening authority. This intended to reduce the trust placed in the opening authority by allowing it to only open signatures on messages permitted by the admitter.

Numerous efficient constructions secure in the above models have been suggested both in the random oracle model [2,10,25,29,30,33,34,42,44,46,57] and in the standard model [1,3,20,21,37,38].

Group Signatures with Revocation. Since revocation is an essential feature of group signatures, different approaches have been proposed for removing members from the group. Bresson and Stern [17] realize revocation by requiring that the signer proves at the time of signing that her group membership credential is not among those contained in a public revocation list. Along this line, Nakanishi et al. [53] gave an efficient scheme (with constant signing and verification times) in the random oracle model. In the standard model, Libert et al. [48,49] gave a number of efficient constructions of group signatures utilizing the subset cover framework [56], originally used in the context of broadcast encryption, to form revocation lists. Another approach, followed by [23,24,28,55,61], uses cryptographic accumulators to incorporate revocation of users: accumulators can be used to give a compact representation of the set of active group members and permit efficient membership proofs.

Boneh et al. [5] showed how to incorporate the removal of users in their initially static group scheme by updating both the group public key and unrevoked members' signing keys. Song [58] also uses key updates to extend the [2] scheme to support revocation and additionally achieve forward security.

Brickell [16] considered a different approach for revocation known as *verifier-local revocation* where the revocation information (i.e., revocation list) is only sent to the verifiers (as opposed to both verifiers and signers) who can check whether a particular signature was generated by a revoked member. This approach was subsequently formalized by Boyen and Shacham [18] and used in, e.g., [45,50,52]. A similar approach is also used in Direct Anonymous Attestation (DAA) protocols [6]. *Traceable Signatures* [41] extend this idea, with a group manager that can release a trapdoor for each member, enabling their signatures to be traced back to the individual user. Releasing tracing information for a member, however, enables the linking of different group signatures by the same revoked member and this by necessity only provides a more relaxed degree of anonymity.

Other variants of group signatures include linkable group signatures [54] and group signatures with controlled linkability [39]. In the former, signatures by the same group member are publicly linkable, whereas in the latter the role of the tracing authority is reduced to the ability of deciding whether two signatures stem from the same (anonymous) member. This approach has been augmented by requiring a proof from the linking authority [11], by specifying a linking authority that obliviously processes linkability

queries across sets of signatures [36] with the results being non-transitive (i.e., signatures are not linkable across queries), or [40] by adding the ability for the opening authority to link as well as open signatures and to also produce proofs of non-authorship. In the latter, users are able to claim or disclaim individual signatures and also link their own signatures.

In this article, our focus is on standard group signatures with strong anonymity where the opening of one signature does not allow the identification of another signature by the same member and thus consider verifier-local revocation and linkable group signatures out of scope.

A generic approach to construct fully dynamic group signatures from accountable ring signatures is suggested in [7] and described in [8]. Bootle et al. [7] also give an efficient instantiation of accountable ring signatures in the random oracle model based on the DDH assumption. In the standard model, Lai et al. [51] give another construction for an accountable ring signature scheme achieving constant size signatures. Following [7, 8], both schemes can be used to obtain fully dynamic group signature schemes with efficiency similar to their ring counterpart.

Security Definitions. The security of static and partially dynamic group signatures has been rigorously formulated [14,19,43,59], and they are now well understood. The security of fully dynamic group signatures is significantly harder to model though. Nakinishi et al. [53] defined security of fully dynamic group signatures in the context of a single group manager that controls group membership and can trace signatures to individual members, and Libert et al. [48] defined security of fully dynamic signatures where a separate opening authority can trace signers. These definitions specifically refer to revocation lists though while a truly general definition of fully dynamic group signatures should in principle admit any mechanism for removing group members. Defining fully dynamic group signatures also leads to subtle questions about when a group member is activated and when a group signature is valid. The formal definitions in [48,53] do in principle, after a revocation has happened, allow the “forging" of signatures for previous epochs. This definitional choice is fine if the application calls for it but implies that existing group signatures become obsolete over time. We will in contrast aim for high generality in our definitions; specifically we want the definitions to be flexible so they can be adapted to any explicit *policy* that specifies when group members should be considered active and governs validity duration of group signatures. Existing definitions for partially dynamic group signatures as well as fully dynamic group signatures assume honestly generated authorities’ keys. To maximize the security guarantees our definitions capture strong security requirements and include, when possible, security against adversarially generated authorities’ keys.

Following our initial conference publication [8], other works adopted and extended our security model. Ling et al. [47] used our model to prove the security of their fully dynamic group signature scheme, which is the first fully dynamic construction based on hardness of lattice assumptions. Our model was recently extended by Backes et al. [12] to include two additional security properties called join and leave privacy, which capture the privacy of users over the time span of their group membership status. El Kaafarani et al. [32] also extended this security model in order to formalize anonymous reputation systems, which allow a set of users to review products anonymously.

1.2. *Our Contribution*

We provide a rigorous security model for fully dynamic group signatures. Despite not assuming a specific design paradigm, in our original version [8] we made a few implicit assumptions regarding the functioning of group signatures. In this revision, we take a step further in modeling fully dynamic group signatures by considerably updating and generalizing [8]. Our model does not preclude current design approaches, and it offers stringent security properties including the features listed below.

- We provide security definitions both for the case of a single group manager and the case where her role is separated from the opening authority. In both settings, we consider, when possible, malicious key generation for the authorities.
- We define a strong correctness property that holds even if the system is populated with malicious users, and in the two-authority setting with a malicious opening authority.
- We define group signatures with respect to an explicit policy for when a user is considered an active group member and for which epochs a group signature is considered valid.
- We consider concurrent joining sessions between the users and the group manager. Existing models restrict the state of the group manager to be compartmentalized on a per user basis resulting in independent joining sessions, where we generalize fully to any choice of manager state between messages in the joining process.
- We formalize two additional security properties called *opening binding* and *opening soundness* capturing that signatures cannot be attributed, respectively, to multiple users, nor to anybody other than the legitimate signer.

To validate our definitions, we show suitable restrictions that relate to the existing definitions for static [14] and partially dynamic [19,43] group signatures. We sketch how the schemes [49,53] based on revocation lists satisfy our definitions or can be modified to satisfy them and also show how the accountable signatures of [7] yield fully dynamic group signatures following a different design paradigm.

2. Definitions for Fully Dynamic Group Signatures

Notation. We write $x \leftarrow S$ for sampling an element x from the set S, where unless otherwise specified we assume the sampling is uniform at random. We write $x \leftarrow A(a)$ for an algorithm A that runs on input a and outputs x. When an algorithm is invoked several times and keeps state between invocations, we may explicitly refer to the state st_A as an additional input and write $x \leftarrow A(a; \mathsf{st}_A)$. A simple example is a single invocation of a probabilistic algorithm that may run in two steps where it first samples randomness r that it stores in its state and then runs the rest of the algorithm $x \leftarrow A(a; r)$ deterministically. We abbreviate deterministic polynomial time DPT and probabilistic polynomial time PPT.

Algorithms may interact with each other. For algorithms A and B, $(x; y) \leftarrow \langle A(a); B(b)\rangle$ denotes the joint execution of A (with input a) and B (with input b) where at the end A outputs x and B outputs y. Delving into the details of the interaction,

it may take place over several rounds during which the algorithms send messages to each other and after which they halt and return their outputs. For an interactive algorithm, a move in the protocol is generated as $(\mathsf{out}; M') \leftarrow A(M)$, where M is the message just received from the other participant or $M = \mathsf{init}$ if this is the first move in the protocol, and M' is the message to send to the other participant. We use the convention that when out is empty ($\mathsf{out} = \varepsilon$), it means that A intends for the interaction to continue, but when out is not empty A will send the last message M' (unless empty) and then terminate the interaction with output out. We note that in the setting of group signatures, the group manager may be involved in multiple concurrent interactions. The group manager's state may therefore change between two rounds in any given joint execution. We write $(\mathsf{out}; M'; \mathsf{st}_A) \leftarrow A(M; \mathsf{st}_A)$ when we explicitly want to indicate the state of an algorithm A may be updated.

We use a security parameter $\lambda \in \mathbb{N}$ to indicate the desired level of security, with the intention that the higher it is, the more secure the scheme should be. In general, when we refer to polynomial time algorithms, we mean that they run in polynomial time in the security parameter and we therefore often give the security parameter to algorithms written in unary 1^λ. We often refer to an adversary $\mathcal{A}$ that is trying to break the system, and when we define security, we usually want the adversary to have negligible probability of breaking the scheme. A function $f : \mathbb{N} \to [0; 1]$ is negligible in the security parameter λ if $f(\lambda) = \lambda^{-\omega(1)}$. For two functions, $f, g : \mathbb{N} \to [0; 1]$ we write $f(\lambda) \approx g(\lambda)$ when $|f(\lambda) - g(\lambda)|$ is negligible. We may therefore simply write $f(\lambda) \approx 0$ to indicate a function is negligible, and conversely, we will refer to a function f as being overwhelming when $f(\lambda) \approx 1$.

We use $\perp$ as an error symbol. Algorithms do not return $\perp$ unless they explicitly want to indicate an error. Conversely, we use the symbol $\top$ to indicate success. We use ε to denote the empty string. We sometimes abbreviate a set $\{1, \ldots, n\}$ as $[n]$.

2.1. *Fully Dynamic Group Signatures*

A fully dynamic group signature ($\mathcal{FDGS}$) scheme involves a set of users who are potential group members and a group manager GM in charge of issuing and revoking group membership. The group signature scheme enables group members to sign messages on behalf of the group in an anonymous way, but in case of abuse the group manager can revoke the anonymity and open the signature to reveal the signer.

We are interested in the fully dynamic setting where users can join and leave the group at any time at the discretion of the group manager. In static or partially dynamic group signatures where members cannot leave it is possible to fix the group information associated with the group at initialization. For a fully dynamic group, however, there has to be a way to prevent a revoked member from using her old key to sign messages. This means the group information associated with the group must change after revocation. We divide the group information into a permanent group public key gpk and temporary group information info_τ, associated with an index τ referred to as an epoch. The group information depicts changes to the group; for instance, it could include the current members of the group (as in accumulator-based constructions) or those who have been excluded from the group (as in constructions based on revocation lists). As

in existing models, we assume that anyone can verify the authenticity of the published group information.

Unlike existing security models for group signatures that assume trusted key generation, we separate key generation from trusted parameter setup. This allows us to define stringent security that protects against adversarial group managers who might generate their keys maliciously. Our definitions can easily be adapted to work for the weaker setting where the group manager's keys are generated honestly as in existing models.

We give two flavors of our definition. We start by providing a definition where the roles of opening signatures and administering group membership are overseen by the same authority and then generalize the definition to the setting where each of those roles is overseen by a separate authority.

2.2. *Syntax*

A fully dynamic group signature scheme $\mathcal{FDGS}$ involves a group manager GM and a set of users. Additionally, there might be the presence of a trusted third party that generates some initial parameters the scheme uses. The scheme consists of the following algorithms and data structures:

- Interactive polynomial time protocol run by GM and a user: **Join**
- Probabilistic polynomial time algorithms: **GSetup**, **GKGen**, **UpdateGroup**, **Sign**, **Open**
- Deterministic polynomial time algorithms: **IsActive**, **Verify**, **Judge**
- Data structure: **Reg**

We will now describe in greater detail the data structure and algorithms and their usage in a $\mathcal{FDGS}$ scheme.

Reg: The registry is a data structure, which is filled as users join the group. The group manager associates any joining group members with session identifiers $\mathsf{i} = 1, 2, 3$, etc. When user i joins, she is able to store a record reg_i in the registry. Once a record is stored, it cannot be changed. The group manager will have read access to the registry and may store the information and use it during opening when tracing the originator of a signature. We model access to the registry with the two following algorithms/oracles:

- $\mathsf{ReadReg}(\mathsf{i})$: On input a session identifier $\mathsf{i} \in \mathbb{N}$, it returns the corresponding entry in the registry reg_i. If no record reg_i is stored, it returns $\perp$.
- $\mathsf{WriteReg}(\mathsf{i}, M)$: On input a session identifier $\mathsf{i} \in \mathbb{N}$ and a message M, it sets $\mathsf{reg}_\mathsf{i} := M$. This oracle can only be used once for every identifier i, further calls with the same session identifier are ignored.

One way to instantiate the registry is with a PKI, which is done explicitly in, e.g., [19]. The registry **Reg** can be hosted by GM but require each entry reg_i to be signed by the user involved in the ith instance of the protocol. Whether one instantiates the registry with a PKI or in a different way is out of scope for this article as long as it gives us the desired functionality.

GSetup$(1^\lambda) \to \mathsf{param}$: There may be a trusted third party that runs this algorithm to generate public parameters param. In case a trusted setup is not required, this algorithm can be regarded as simply setting $\mathsf{param} := 1^\lambda$.
GKGen$(\mathsf{param}) \to (\mathsf{out}_{\mathsf{GM}}; \mathsf{st}_{\mathsf{GM}})$: The group manager uses this algorithm to generate $\mathsf{out}_{\mathsf{GM}} := (\mathsf{mpk}, \mathsf{info}_0)$ consisting of the manager's public key and the initial group information, and the resulting state $\mathsf{st}_{\mathsf{GM}}$ of the group manager. The group public key is $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk})$.
Join: To enroll a user as a member, the GM may run the interactive joining protocol with her. Their respective algorithms are:

- $\mathbf{Join}_{\mathsf{User}}^{\mathsf{WriteReg}(\mathsf{i},\cdot)}(M; \mathsf{st}) \to (\mathsf{out}; M_{\mathsf{GM}}; \mathsf{st})$: This algorithm specifies the user's execution of the interactive joining protocol with GM. Given an input message from GM and the user's internal state st, it returns a message for GM and a new state. In its first call, the algorithm is executed on initial input $(\mathsf{init}; \mathsf{gpk})$. In each instance of the protocol, the user is allowed a single call to the oracle $\mathsf{WriteReg}(\mathsf{i}, \cdot)$ for writing into the registration table an entry reg_i corresponding to its identifier i. Joining session i terminates after at most $k(\lambda)$ rounds by a call returning $(\mathsf{gsk}; M_{\mathsf{GM}}; \mathsf{st})$, which includes the user's secret key $\mathsf{gsk}_\mathsf{i} := \mathsf{gsk}$, an optional final message for the issuer including a termination message done, and the user final state. If it terminates with $\mathsf{gsk} = \perp$, the user will consider it as a fail to join, and on failure it will always be the case that it ends with $M_{\mathsf{GM}} = (\mathsf{done}, \perp)$. After termination, the user will ignore all future inputs to $\mathbf{Join}_{\mathsf{User}}$.
- $\mathbf{Join}_{\mathsf{GM}}^{\mathsf{ReadReg}(\mathsf{i})}(\mathsf{i}, M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{GM}}) \to (\mathsf{out}_{\mathsf{GM}}; M; \mathsf{st}_{\mathsf{GM}})$: This algorithm specifies GM's execution in the interactive joining protocol with a user. The manager GM keeps track of distinct instances of the protocol using unique identifiers i, which we without loss of generality assume are numbered 1, 2, 3, etc. The algorithm receives as input a session identifier i, a message M_{GM} received from the user, and the GM's internal state, and it returns a message M for the user interacting in session i and updates the state $\mathsf{st}_{\mathsf{GM}}$. The algorithm has access to the oracle $\mathsf{ReadReg}(\mathsf{i})$ to read the entry reg_i in the registration table $\mathbf{Reg}$.[1] Each joining session will terminate after at most a polynomial number of rounds. We let $k(\lambda)$ be the maximal number of rounds before termination. Termination will be indicated in the local output $\mathsf{out}_{\mathsf{GM}}$ of the manager GM and can be successful $(\top)$ or fail $(\perp)$, and if it fails, the output message will be $M_\mathsf{i} = (\mathsf{done}, \perp)$. After termination, GM ignores future calls with the same i.

For conciseness, we will often refer to the user involved in the ith session of the **Join** protocol with the manager as user i. We note that the user may not be aware of her own session identifier i, since she may not be aware of how many other users are joining or have already joined the group.
UpdateGroup$(\mathcal{R}; \mathsf{st}_{\mathsf{GM}}) \to (\mathsf{info}; \mathsf{st}_{\mathsf{GM}})$: The group manager runs this algorithm to update the group information, where the set $\mathcal{R}$ consists of session identifiers

[1] We note that after reading reg_i during a join session, the group manager can opt to store it in $\mathsf{st}_{\mathsf{GM}}$. The group manager may therefore be aware of other entries in the registry and could even build its own internal copy of a full $\mathsf{ReadReg}(\cdot)$ oracle if it is willing to spare the storage.

associated with users to be revoked. The algorithm returns new public group information info and updates the state of GM. The group information info may or may not depend on the set of newly joined members of the group, which the group manager records in its internal state. The group information info is intended as group information pertaining to the group, and we will in general assume anybody may have access to the sequence info_0, info_1, etc.; the group manager creates during the lifetime of the group signature scheme.

IsActive$(\mathsf{i}, \tau, \mathsf{st}_{\mathsf{GM}}) \rightarrow 1/0$: The **Join** protocol and the **UpdateGroup** algorithm describe how an honest GM adds and revokes group members. The exact moment when a member is activated and able to sign is design specific. In some constructions, group members are implicitly activated after successfully terminating the **Join** protocols and may even be able to sign with respect to previous epochs; in others, they are explicitly activated by GM when a new group information info_τ is published. Consequently, different design choices lead to different time spans when members are allowed to sign. In order to take into account these differences in the security definitions without favoring a particular design paradigm, we use the **IsActive** procedure, which should be interpreted as the group manager's policy for when a member is considered active.

The **IsActive** algorithm takes as input a session identifier i, an epoch τ associated with group information info_τ the group manager has published earlier, and the state of the group manager $\mathsf{st}_{\mathsf{GM}}$. We refer to a user as an *active* member of the group at epoch τ if and only if the algorithm returns 1. We place the following constraints on the policies an honest group manager can have for when a user is active:

- If τ is not associated with any info_τ the group manager has published, the algorithm returns 0.
- If i is not associated with a joining session where the group manager has terminated successfully, the algorithm returns 0.
- If i was revoked when creating info_τ for this epoch or earlier, the algorithm returns 0.
- If i is associated with a joining session where the group manager ended her part successfully before info_τ was created, and user i is not revoked at or before epoch τ, the algorithm returns 1.

Sign$(\mathsf{gsk}, \mathsf{info}, m) \rightarrow \Sigma$: Given a user's group signing key gsk, group information info, and a message m, the signing algorithm outputs a group signature Σ.[2]

Verify$(\mathsf{gpk}, \mathsf{info}, m, \Sigma) \rightarrow 1/0$: The verification algorithm checks whether Σ is a valid group signature on m with respect to the group information info and outputs a bit: 1 for accept and 0 for reject.

Open$(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, \mathsf{info}, m, \Sigma) \rightarrow (\mathsf{i}, \pi)$: The opening algorithm receives as input the group public key gpk, the state of the group manager, some public group information info, a message, and a signature. It returns a session identifier i together with a proof π attributing Σ to user i. If the algorithm is unable to attribute the signature

[2] We note as a special case that the signing algorithm may be such that it completely ignores info, which potentially may make it faster. This speed advantage is similar to the efficiency the signer may have in group signatures with verifier-local revocation where only verifiers receive information about which members are revoked.

to a particular group member, it returns $(\perp, \pi)$ to indicate that it could not attribute the signature.

Judge$(\mathsf{gpk}, \mathsf{info}, \mathsf{reg}, m, \Sigma, \pi) \rightarrow 1/0$: The judge algorithm checks the validity of a proof π attributing the signature Σ on m w.r.t. group information info to a user with registry record reg. It outputs 1 for accept and 0 for reject.

Separating the Role of Group Manager and Opening Authority. Following the suggestion of Camenisch and Michels [26], Bellare et al. [19] separate the group manager role we described above into two parts: a group manager GM (who they call the Issuer) administrating group membership and an opening authority OA (who they call the Opener) capable of tracing the signer of a message. There are natural settings where such a division of roles may be called for; an organization may, for instance, consider group management the task of the human resources department and opening signatures the domain of the fraud department. While the separation of the group manager and opening authority roles complicates definitions a little, they also have the advantage of permitting more fine-grained security notions that allow for adversarial behavior in either the group manager or the opening authority.

When separating out the role of the opening authority, the syntax of a group signature scheme changes. Key generation can now be seen as a joint process that involves both the group manager and the opening authority. Sometimes it may be desirable to minimize interaction and allow authorities to generate their own keys independently, but for maximal generality we define key generation as an interactive protocol between them, where independent key generation is a special case. Another change is that since the opening algorithm is run by the opening authority, it needs access to read the registry to know who to attribute a given signature to. We describe these changes below:

GKGen: To generate the group public key, GM and OA may run an interactive protocol. Their respective algorithms are:

- $\mathbf{GKGen}_{\mathsf{GM}}(M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{GM}}) \rightarrow (\mathsf{out}_{\mathsf{GM}}; M_{\mathsf{OA}}; \mathsf{st}_{\mathsf{GM}})$: This algorithm specifies the GM's execution in the interactive key generation protocol with OA. It gets as input a message M_{GM} received from OA and the GM's internal state and returns an output $\mathsf{out}_{\mathsf{GM}}$, a message M_{OA} for OA and updates the state to $\mathsf{st}_{\mathsf{GM}}$. The state of the group manager is initialized as $\mathsf{st}_{\mathsf{GM}} := \mathsf{param}$. If GM initiates the protocol, the input message is initialized as $M_{\mathsf{GM}} := \mathsf{init}$. In a successful execution of the protocol, the last call of the algorithm returns a non-empty output value $\mathsf{out}_{\mathsf{GM}} := (\mathsf{mpk}, \mathsf{info}_0)$ consisting of the GM's public key and the initial group information. After termination, subsequent calls to the algorithm will be ignored.
- $\mathbf{GKGen}_{\mathsf{OA}}(M_{\mathsf{OA}}; \mathsf{st}_{\mathsf{OA}}) \rightarrow (\mathsf{out}_{\mathsf{OA}}; M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{OA}})$: This algorithm specifies the OA's execution in the interactive key generation protocol with GM. It gets as input a message from GM and the OA's internal state and it returns an output $\mathsf{out}_{\mathsf{GM}}$, a message M_{GM} for GM and updates the state $\mathsf{st}_{\mathsf{OA}}$. The state of OA is initialized as $\mathsf{st}_{\mathsf{OA}} := \mathsf{param}$. If OA initiates the protocol, the input message is initialized as $M_{\mathsf{OA}} := \mathsf{init}$. In a successful execution of the protocol, the last call of the algorithm returns an non-empty output value $\mathsf{out}_{\mathsf{OA}} := (\mathsf{opk}, \mathsf{osk})$ consisting of the OA's public and secret keys. Subsequent calls of the algorithm are ignored.

We denote an entire execution of the key generation protocol as

$$((\mathsf{mpk}, \mathsf{info}_0; \mathsf{st}_{\mathsf{GM}}); (\mathsf{opk}, \mathsf{osk})) \leftarrow \langle \mathbf{GKGen}_{\mathsf{GM}}(\mathsf{param}); \mathbf{GKGen}_{\mathsf{OA}}(\mathsf{param}) \rangle$$

and let $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$ be the group public key.

Open$^{\mathsf{ReadReg}(\cdot)}(\mathsf{gpk}, \mathsf{osk}, \mathsf{info}, m, \Sigma) \rightarrow (\mathsf{i}, \pi)$: The opening algorithm receives as input the group public key gpk, the opening key osk, group information info, a message, and a signature. It returns a session identifier i together with a proof π attributing Σ to user i. If the algorithm is unable to open the signature to a particular group member, it returns $(\perp, \pi)$ to indicate that it could not attribute the signature.

Relation Between Definitions with Single and Separate Authorities. Group signatures with separate and without separate authorities are closely related. Specifically, given a group signature scheme $\mathcal{FDGS}$ for separate GM and OA, we can define a single authority group signature scheme $\mathcal{FDGS}'$, where the group manager GM' first runs the interaction $((\mathsf{mpk}, \mathsf{info}_0; \mathsf{st}_{\mathsf{GM}}); (\mathsf{opk}, \mathsf{osk})) \leftarrow \langle \mathbf{GKGen}_{\mathsf{GM}}(\mathsf{param}); \mathbf{GKGen}_{\mathsf{OA}}(\mathsf{param}) \rangle$ and returns the manager public key $\mathsf{mpk}' := (\mathsf{mpk}, \mathsf{opk})$ and sets the state to be $\mathsf{st}'_{\mathsf{GM}} := (\mathsf{st}_{\mathsf{GM}}, \mathsf{osk})$. Whenever a new user joins the group, GM has access to an oracle that allows it to read the corresponding record in the registry, and we imagine GM' keeps track of these registry entries so it has its own virtual $\mathsf{ReadReg}$ oracle. Whenever GM' has to run the opening algorithm, it will then just use osk. All other algorithms in $\mathcal{FDGS}'$ are defined in the natural way from $\mathcal{FDGS}$. It is easy to see that this transformation preserves the efficiency of $\mathcal{FDGS}$ and we will in the following argue that security is preserved as well.

2.3. *Security Definitions*

In the following, we will generally use the index h to refer to the honest user in security games where she is unique and $\mathcal{H}$ to refer to the set of honest users, as appropriate. We will use ⌜dashed⌝ text when referring to the separate authority setting. As such, dashed text should be ignored in the single authority setting.

Definition 1. An $\mathcal{FDGS}$ with the syntax in Sect. 2.2 is a fully dynamic group signature if it is correct, anonymous, traceable, and non-frameable as defined below.

Correctness. Correctness guarantees that an honest user can enroll in the group and produce signatures that are accepted by the **Verify** algorithm. We assume in the correctness definition that GM is honest and willing to enroll the user, since otherwise it could just refuse membership. However, there may be other users that are malicious. We therefore model correctness as an adversarial game, where the adversary acts on behalf of all other users and we want even in this setting that the honest user can successfully enroll and sign messages. The correctness definition captures three aspects in this setting:

- An honest user interacting with an honest GM should be able to enroll in at most $k(\lambda)$ rounds after which the user terminates successfully with a key $\mathsf{gsk}_{\mathsf{i}}$.
- The group manager should before or in the same round terminate with success indicator $\top$ and activate the user no later than the next update.

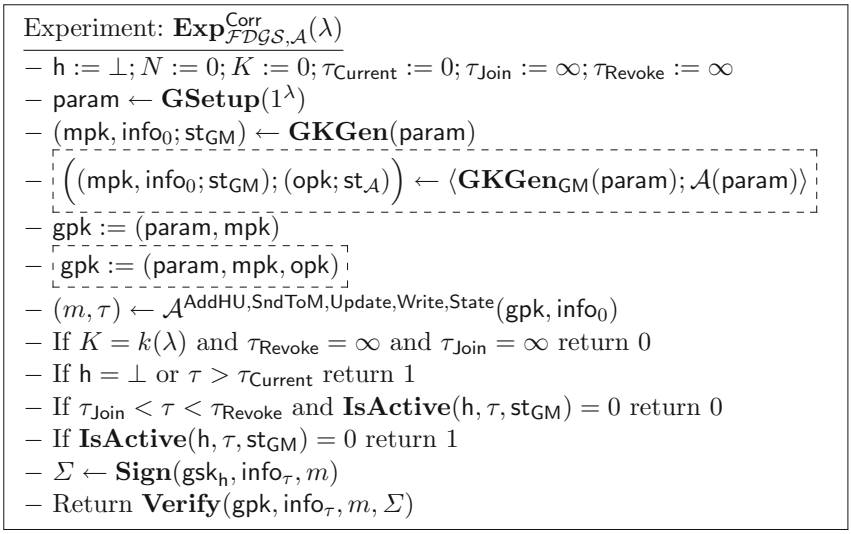

Fig. 1. Correctness game. Dashed text is omitted in the single authority setting, whereas in the separate authorities setting Dashed text replaces the preceding line.

- Once activated, the user should be able to sign messages.

These three properties should hold as long as the user is not revoked.

In the game $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{FDGS},\mathcal{A}}$ (shown in Fig. 1), we grant the adversary $\mathcal{A}$ access to the following oracles, details of which are given in Fig. 2. We maintain several global counters: h is the index of the joining session initiated by the honest user, N is the number of users that initiated the **Join** protocol with GM, and τ_{Current}, τ_{Join}, τ_{Revoke} is the current epoch, the epoch during which the user created her key, τ_{Revoke} is the time the user was revoked (if ever), and K is the number of calls to the AddHU oracle, i.e., the number of rounds executed by the honest user in the **Join** protocol.

AddHU(): This oracle adds a single honest user to the group. Each call of the oracle executes the next round of interaction in the **Join** protocol between the honest user and the honest group manager. It returns the exchanged messages as well as the outputs of both parties. Note that the adversary learns the group signing key of the honest user at the successful conclusion of the interaction.

SndToM(i, M_{GM}): This oracle allows the adversary to add a corrupt user to the group. The adversary can deviate from the **Join** protocol by sending arbitrary messages M_{GM} to GM. Each oracle call executes the next move of an honest GM on input message M_{GM} in the ith instance of the **Join** protocol. It returns the GM's response message and output.

Update($\mathcal{R}$): This oracle allows the adversary to update the public group information. Here $\mathcal{R}$ is the set of the group members to be removed from the group. Calling this oracle triggers a new epoch.

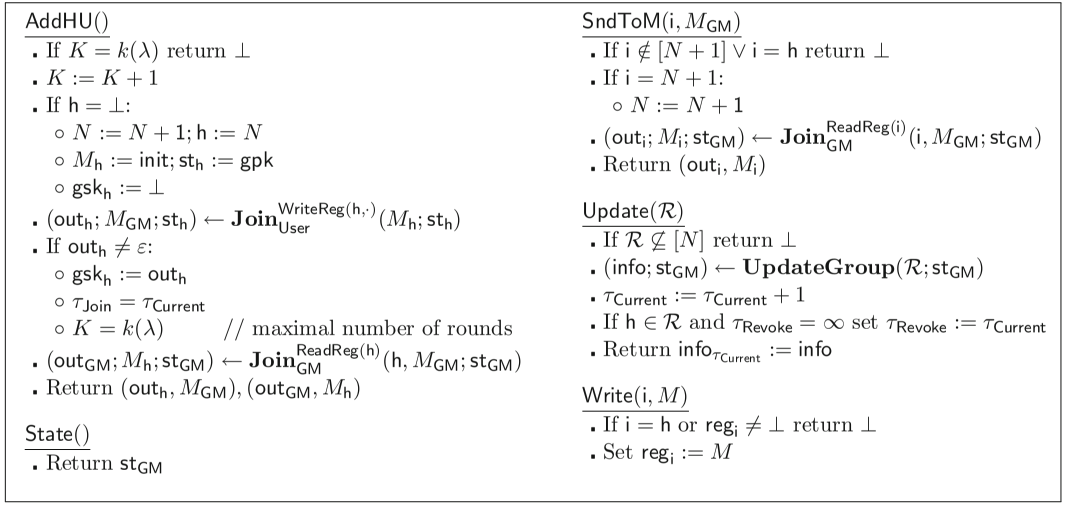

Fig. 2. Oracles used in the correctness game.

Write(i, M): Given a session identifier and a message M, the oracle sets $\mathsf{reg}_i := M$. The oracle can only be used once for every identifier i, further calls to it return $\perp$ without producing changes to the registry. It cannot be called on the identifier corresponding to the joining session initiated by AddHU.

State(): This oracle returns the current GM's state $\mathsf{st}_{\mathsf{GM}}$.

In the correctness definition, we restrict the adversary to enrolling a single honest user into the group via calling the AddHU oracle. This generalizes to the case of multiple honest users via a standard hybrid argument.

Definition 2. (*Correctness*) An $\mathcal{FDGS}$ scheme is *correct* if for any PPT adversary $\mathcal{A}$

$$\Pr[\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1] \approx 1$$

If the definition holds also for unbounded adversaries, we say the $\mathcal{FDGS}$ scheme is statistically correct, and if the probability is exactly 1, we say the $\mathcal{FDGS}$ scheme is perfectly correct.

Variations in Correctness. In the correctness definition, we give the adversary access to the state of the group manager. This means even if the honest group manager's secret data are leaked, an honest user can still enroll and sign messages as long as the group mr considers her active. A more relaxed but still reasonable definition of correctness would be to assume the group manager keeps her state secret. In this latter case, it is then natural to give the adversary access to an opening oracle to model that the group manager may sometimes trace a member who produced a signature.

Many definitions of correctness found in the literature [14,19,43] encompass not just that an honestly generated signature is accepted by the verification algorithm but also add other requirements such as an honestly generated signature should be opened to the honest signer who generated it. In our definition of correctness, we only require that honestly generated signatures are accepted and refer to other security definitions to handle additional requirements that we consider less central. In particular, for the property of honestly generated signatures opening to the correct signer, it is captured by the traceability and opening soundness properties we later define, and captured in a much stronger sense than usually done in correctness definitions since we explicitly consider opening soundness in a highly adversarial setting instead of just considering an honest interaction. This leaves a small definitional gap when considering *perfect* correctness since traceability and opening soundness may only hold computationally. However, we find the difference to be insignificant and have therefore deliberately opted for the minimal and simplest definition of correctness that only demands honest generated signatures to be accepted.

Correctness Under Separate Authorities. In the case where there is a separation between the group manager and an opening authority, we still need the group manager to be honest for correctness to make sense. However, we may want correctness to hold even in the presence of a malicious opening authority, since it is the sovereign domain of the group manager to decide who is an active member and should be able to sign messages. In the dashed version of Fig. 1, we therefore define the correctness game with an adversarial

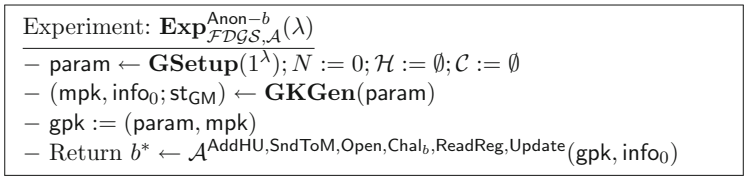

Fig. 3. Anonymity game.

opening authority. It is straightforward to see that given a group signature scheme $\mathcal{FDGS}$ with separate GM and OA satisfying correctness, the transformation from Sect. 2.2 gives a group signature scheme $\mathcal{FDGS}'$ with a single group manager that is correct too.

Anonymity. Group signatures should be anonymous and not reveal the identity of the group member who produced them. Since the group manager has the ability to trace signers, we must assume the group manager to be honest for anonymity to hold, but some of the other users may be malicious. We will be relaxing this assumption when we consider the case of separate authorities in the next paragraph.

In the game $\mathbf{Exp}^{\mathsf{Anon}-\mathsf{b}}_{\mathcal{FDGS},\mathcal{A}}$ (shown in Fig. 3), we maintain the following counter and lists: N is the number of users that initiated the **Join** protocol with GM, $\mathcal{H}$ is a list of honest users, and $\mathcal{C}$ is a list of challenge signatures obtained from the challenge oracle. We give the adversary access to the following oracles, details of which are given in Fig. 4.

AddHU(i) : This oracle allows the adversary to add honest users to the group by going through the join protocol one round at a time. The oracle models full key exposure; both communication and the user's signing key are leaked to the adversary.

SndToM(i, M_{GM}) : This oracle allows the adversary to add corrupt users to the group. The adversary can deviate from the **Join** protocol by sending arbitrary messages M_{GM} to GM. Each oracle call executes the next move of an honest GM on input message M_{GM} in the ith instance of the **Join** protocol. It returns the GM's response message and output.

$\mathsf{Chal}_b(\mathsf{info}, m, \mathsf{i}_0, \mathsf{i}_1)$: This a left–right oracle for defining anonymity. It takes as input some group information info, a message m, and two honest users $\mathsf{i}_0, \mathsf{i}_1$. It returns a group signature on the message using key $\mathsf{gsk}_{\mathsf{i}_b}$ for $b \leftarrow \{0, 1\}$ and the given group information. It is required that both challenge users are able to sign with respect to info. The adversary can only call this oracle once.

Open(info, m, Σ) : Returns the session identifier i of the signer who produced signature Σ on m with respect to info, together with a proof π. The oracle cannot be called on a signature obtained from the Chal_b oracle.

ReadReg(i) : Given a session identifier i, it returns the corresponding entry in the registry reg_i.

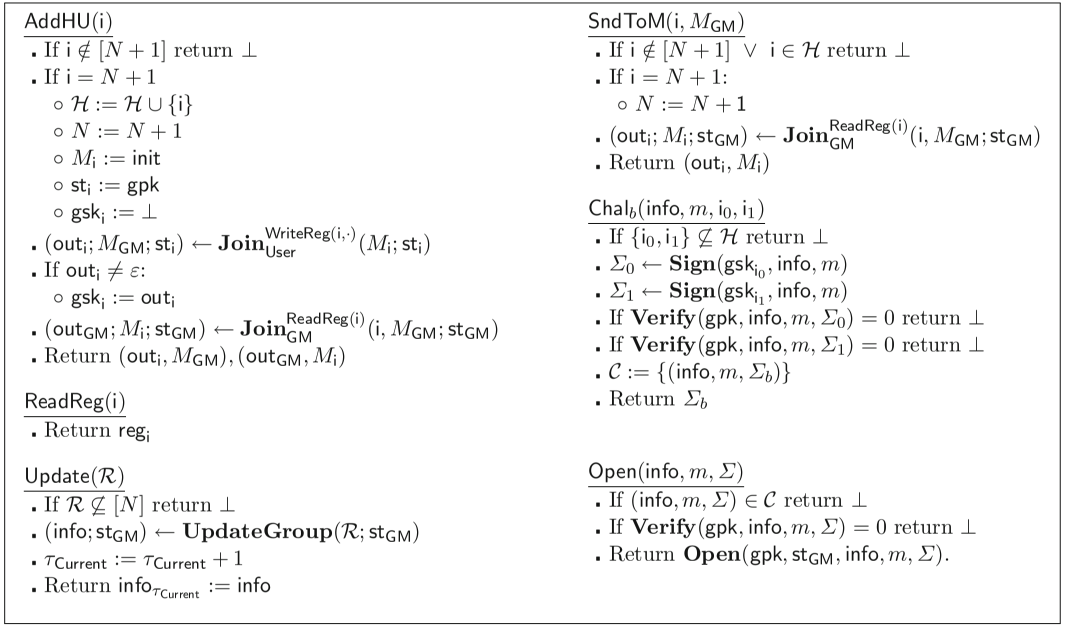

Fig. 4. Oracles used in the anonymity game.

$\mathsf{Update}(\mathcal{R})$: Allows the adversary to prompt a group information update and increment the epoch. Here $\mathcal{R}$ is a set of the group members to be revoked from the group.

The adversary can interact with honest users and join corrupt users. At some point, the adversary picks two honest members of the group at a chosen epoch, gets a signature from one of them, and tries to learn which of them has signed a chosen message. She wins if she can guess which member signed the message. For simplicity, the adversary is only allowed a single challenge query, but a standard hybrid argument (similar to that used in [19]) shows this is equivalent to seeing many challenge signatures. Our definition covers full key exposure attacks by allowing $\mathcal{A}$ to learn the secret keys of all users in the group.

Definition 3. (*Anonymity*) An $\mathcal{FDGS}$ scheme is *anonymous* if for all PPT adversaries $\mathcal{A}$ the following advantage is negligible

$$\mathbf{Adv}^{\mathsf{Anon}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) := \left|\Pr[\mathbf{Exp}^{\mathsf{Anon}-0}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Exp}^{\mathsf{Anon}-1}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1]\right|.$$

Variations in Anonymity. Our definition of anonymity corresponds to what Bellare et al. [14] call full anonymity. Their usage of *full* anonymity emphasizes that anonymity holds even in the presence of an adversary that sees the signing keys of honest users and has access to an opening oracle. Our definition captures full anonymity, as the adversary sees not only the signing key of every honest user, but she also sees the entire joining transcript. Full anonymity gives strong security guarantees since it ensures that even if a user's secret key is leaked her past or future signatures still do not reveal her identity.

Group signatures with full anonymity imply the existence of IND-CCA secure public-key encryption [4,23]. The anonymity notion therefore has to be relaxed if we want to build group signatures based on one-way functions as is done in [23]. Such a relaxation can consist in not giving out_i to the adversary in the AddHU oracle but instead give the adversary access to a signing oracle that will allow it to get signatures from honest users on any message of its choosing.

Boneh et al. [5] define another relaxed form of anonymity where the adversary does not have access to the Open oracle. This relaxation is analogous to the distinction between IND-CPA and IND-CCA secure public-key encryption, and indeed, they refer to the notion as CPA-anonymity.

Anonymity Under Separate Authorities. Let us now consider the case where we separate the roles of managing the group, GM, and the role of opening signatures, OA. For anonymity to hold, we need the opening authority to be honest; however, we may desire security against a malicious group manager. We give the corresponding anonymity game in Fig. 5. We observe that the oracles the adversary has access to are different because when the adversary runs the group manager it can directly manage the joining interaction with honest users, simulate the enrollment of corrupt users, and compute group information updates by itself. Therefore, we remove the Update, SndToM oracles and replace the AddHU oracle with an SndToU oracle that lets the adversarially controlled group manager communicate with an honest user trying to join the group.

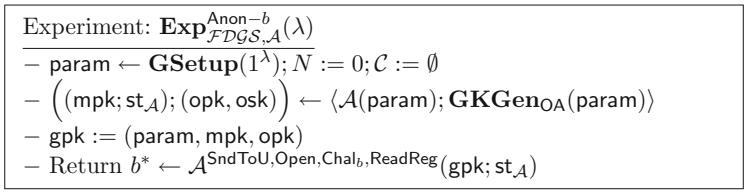

Fig. 5. Anonymity game for separate GM and OA.

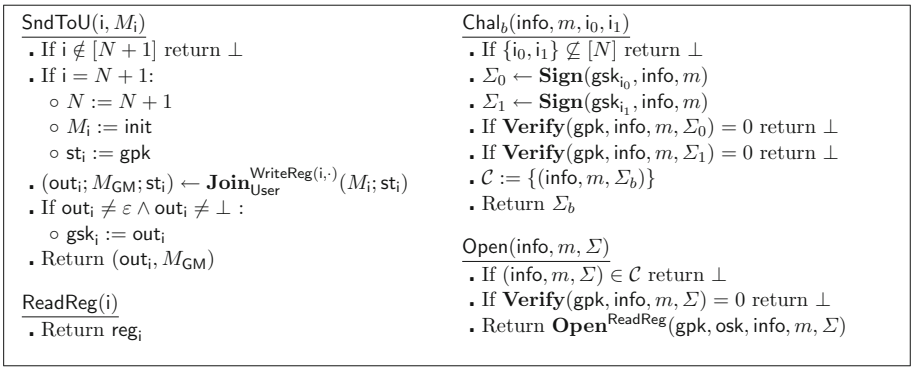

Fig. 6. Oracles used in the anonymity game for separate GM and OA.

Let $\mathcal{FDGS}$ be a group signature scheme with separate GM and OA with full anonymity, and let $\mathcal{FDGS}'$ be the resulting single authority group signature scheme resulting from the transformation given in Sect 2.2. Then, $\mathcal{FDGS}'$ is anonymous according to the single authority definition because an $\mathcal{FDGS}$ adversary can as a special case run an honest GM algorithm and simulate everything that happens in an attack against $\mathcal{FDGS}'$. In particular, by running GM honestly, it can simulate the SndToM oracle and use the SndToU oracle to build a simulated AddHU oracle (Fig. 6).

Traceability. Traceability protects the group manager by ensuring that all signatures that are valid for a given epoch can be opened to an active member of the group. In the traceability game $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{FDGS},\mathcal{A}}$ shown in Fig 7, we maintain a counter N for the number of users that initiated the **Join** protocol with GM. The adversary $\mathcal{A}$ has access to the following oracles, details of which are given in Fig. 8.

SndToM(i, M_GM) : This oracle allows the adversary to add corrupt users to the group. She can deviate from the **Join** protocol by sending arbitrary messages M_GM to GM. Each oracle call executes the next move of an honest GM on input message M_GM in the ith instance of the **Join** protocol. It returns the GM's output and response message.

Update($\mathcal{R}$) : This oracle allows the adversary to trigger a group information update and increment the epoch. Here $\mathcal{R}$ is the set of the group members to be removed from the group.

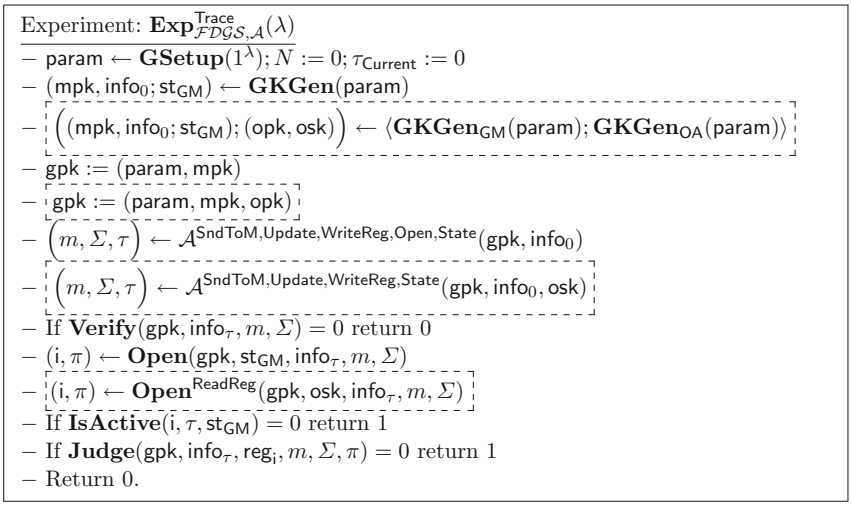

Fig. 7. Traceability game. Dashed text is omitted in the single authority setting, whereas in the separate authorities setting Dashed text replaces the preceding line.

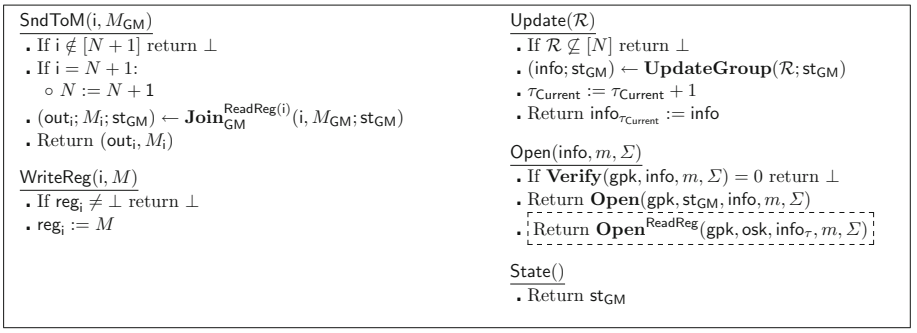

Fig. 8. Oracles used in the traceability game. Dashed text is omitted in the single authority setting, whereas in the separate authorities setting Dashed text replaces the preceding line.

WriteReg(i, M) : Given a session identifier and a message M, the oracle sets $\mathsf{reg}_\mathsf{i} := M$. The oracle can only be used once for every identifier i.

Open(info, m, Σ) : Returns the session identifier i of the signer who produced signature Σ on m with respect to info, together with a proof π.

Definition 4. (*Traceability*) An $\mathcal{FDGS}$ scheme is *traceable* if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible

$$\mathbf{Adv}^{\mathsf{Trace}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1].$$

Variations in Traceability. Bellare et al. [14] defined traceability purely with respect to identifying a signer but did not require a proof of correct opening. Bellare et al. [19] included the use of a proof for correct opening that can be verified by anybody using the **Judge** algorithm. If we trust the group manager instead of requiring a proof of correct opening, the game in Fig. 7 can be simplified by eliminating the proof and the **Judge** algorithm.

By necessity, the group manager needs to be at least partially trusted since otherwise it can just enroll some dummy member that can then sign arbitrary messages and act as a scapegoat. However, we have defined traceability such that it holds even if the honest group manager's secret state is leaked. A reasonable relaxation of our definitions would be to trust the group manager to keep its state secret, in which case we would remove the State oracle.

Traceability Under Separate Authorities. Let us consider the case where we separate the roles of group manager and opening authority. The opening authority is the primary stakeholder that wants to ensure signers can be traced. However, we define security strongly by requiring traceability even in case its secret opening key is leaked. As in the single authority setting, we still need to have some trust in the group manager to keep track of who is active in the group and not to enroll dummy members, however, again we opt for a strong definition of security where its state may be leaked.

The two games for traceability are very similar, and it is easy to see that the transformation in Sect. 2.2 of an $\mathcal{FDGS}$ for separate GM and OA to a single group manager scheme $\mathcal{FDGS}'$ preserves traceability.

Non-frameability. Non-frameability is a security notion that says even if the rest of the group as well as the group manager are fully corrupt, they cannot falsely attribute a signature to an honest member who did not produce it. In the non-frameability game $\mathbf{Exp}^{\mathsf{Non-Frame}}_{\mathcal{FDGS},\mathcal{A}}$ shown in Fig. 9, we grant the adversary access to the oracles described below and detailed in Fig. 10, and we keep a global list $\mathcal{S}$ of signatures produced by an honest user. We note that the adversary controls the group manager and hence session identifiers no longer carry much meaning the adversary can pretend the user has any session identifier. Instead, without loss of generality we simply identify the honest user with a generic record reg and require that only the honest user is able to write in this record.

SndToHU(M_{h}) : This oracle allows the adversary to interact with a single honest user in an instance of the join protocol. Each call executes the next move of the honest user on input a message M_{h} provided by the adversary (playing the role of the corrupt group manager) and returns the user response message. The user may write a message into the register using oracle Write, which can be accessed only once by the user and reveals the register entry to the adversary. The user output $\mathsf{out}_{\mathsf{h}}$, i.e., the signing key, is *not* disclosed to the adversary.

SignHU(info, m) :This oracle is used by the adversary against non-frameability to obtain signatures from an honest group member h added to the group via SndToHU calls. It returns a group signature on the message m using key $\mathsf{gsk}_{\mathsf{h}}$ and group information info.

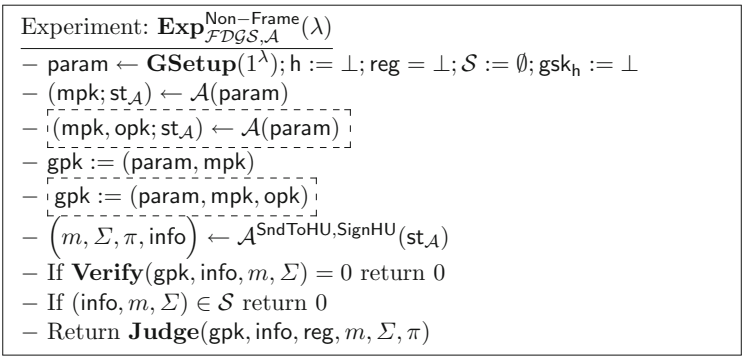

Fig. 9. Non-frameability game. Dashed text is omitted in the single authority setting, whereas in the separate authorities setting Dashed text replaces the preceding line.

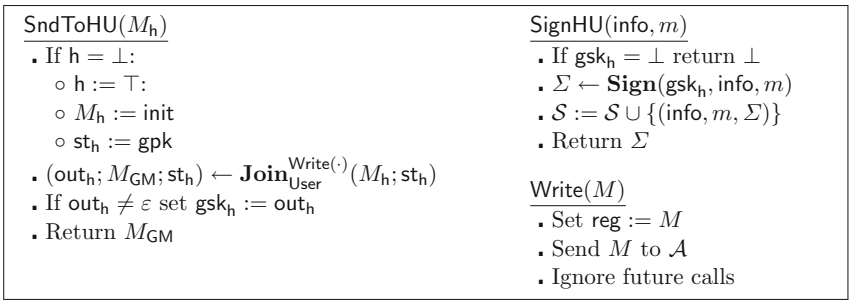

Fig. 10. Oracles used in the non-frameability game.

Definition 5. (*Non-frameability*) An $\mathcal{FDGS}$ scheme is *non-frameable* if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible

$$\mathbf{Adv}^{\mathsf{Non-Frame}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{Non-Frame}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1].$$

In our definition of non-frameability, the adversary controls the group manager during the key generation process. Thus, our definition is stronger than existing definitions, which only allow the group manager to be corrupted after the group keys have been honestly generated.

We allow only a single honest user in the group and ask the adversary to frame her. It can be shown that this implies the more general case involving several honest users in the group by a standard hybrid argument since the adversary can simulate the actions of additional honest users.

Variations in Non-frameability. While traceability still makes sense without proofs and the **Judge** algorithm, non-frameability does not since it is about whether a corrupt group manager would be able to *prove* instead of just falsely accusing an honest user of having signed a message. If we consider the setting with no proofs and no **Judge** algorithm, we can therefore completely eliminate the non-frameability notion from our definitions.

Non-frameability Under Separate Authorities. In the dashed version of Fig. 9, we give a variation in the non-frameability game suitable for the setting where the roles of group manager and opening authority are separated. Since both are under adversarial control, the game is equivalent to the previous non-frameability game where the group manager has both roles, so it is easy to see the transformation from Sect. 2.2 of a two-authority $\mathcal{FDGS}$ into a single authority $\mathcal{FDGS}'$ preserves non-frameability.

2.4. *Additional Security Definitions*

In addition to the core security properties correctness, anonymity, traceability, and non-frameability, a group signature scheme may have additional security guarantees. In the above definitions, the opening provided by either the group manager or the opening authority can be generally thought of as a deterrent against members misbehavior. It is not hard, however, to envision applications in which openings could be used to positive benefit for the members. In such scenarios, it may become crucial to prevent an attacker exploiting the opening mechanism to her own advantage rather than to elude it.

Opening Binding. Opening binding[3] defined by Sakai et al. [59] in the context of partially dynamic group signatures guarantees that even if all authorities and users collude they should not be able to produce a valid signature that can be selectively attributed to different members. Consider, for instance, a contest to find the best stock market analyst. Group signatures are used to sign stock market predictions by the experts, who should remain anonymous in order not to influence the markets, and later, we use the opening algorithm in order to tally up who is the best expert. There may be a financial incentive to become the leading expert, so we could imagine a collusion where an “expert" and a dummy “novice" enroll and then collaborate to attribute all correct predictions to the “expert" and all wrong predictions to the “novice."

We describe the opening binding game in Fig. 11, where the goal of the adversary is to create a signature and two distinct attributions to who signed it. We consider a strongly adversarial setting, where both the authorities and users may be adversarial but want the guarantee that each signature must be attributed to a unique record in the registry.

Definition 6. (*Opening Binding*) An $\mathcal{FDGS}$ scheme is *opening binding* if for all PPT adversaries $\mathcal{A}$

$$\mathbf{Adv}^{\mathsf{Opening-Bind}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{Opening-Bind}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1] \approx 0.$$

[3] Sakai et al. [59] refer to this notion as opening soundness.

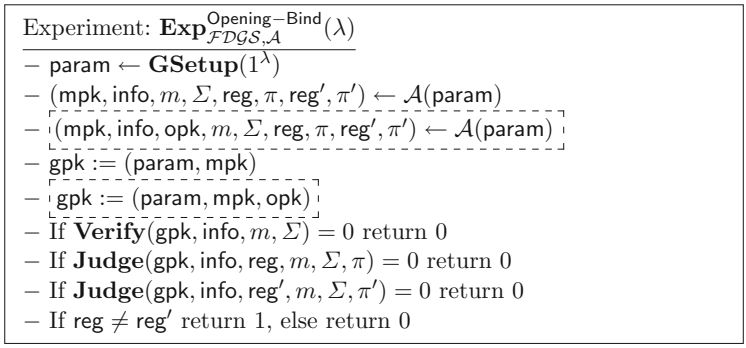

Fig. 11. Opening binding game. Dashed text is omitted in the single authority setting, whereas in the separate authorities setting Dashed text replaces the preceding line.

Opening Soundness. Consider again the example of a competition to determine who is a the best stock market prediction expert, this time from the perspective of an honest expert. It would be problematic if the opening of her signature did not point to herself but instead attributed it to somebody else. The worst-case scenario here is that dishonest authorities are collaborating with a malicious user to attribute an honest user's signature to a malicious user instead.[4]

We define the opening soundness experiment in Fig. 12, where the adversary is trying to attribute a signature Σ of an honest user to a different registry. The experiment uses an oracle to enroll an honest user and an oracle that provides an honestly generated signature described in Fig. 13.

Definition 7. (*Opening Soundness*) An $\mathcal{FDGS}$ scheme is *opening sound* if for all PPT adversaries $\mathcal{A}$

$$\mathbf{Adv}^{\mathsf{Opening-Sound}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{Opening-Sound}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1] \approx 0.$$

Related notions of opening soundness were previously considered by Kiayias and Yung [43], as a requirement for correctness, and Sakai et al. [59], under the name of *weak* opening soundness. Our definition considers highly adversarial settings and thus captures a much more stringent notion.

[4] We observe a subtle difference between opening binding and opening soundness. Opening binding protects against malicious actors trying to create double openable signatures, while opening soundness guarantees an honest signer can indeed have the signature attributed to her. Neither definition implies the other. It is conceivable one could violate opening binding by creating two distinct openings of the same valid but maliciously generated signature without violating the attribution of honest signatures to honest signers. It is also conceivable that honest signatures can only be opened in one way but with a malicious setup that unique attribution does not point to the honest signer.

Experiment: $\mathbf{Exp}^{\mathsf{Opening-Sound}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$

- $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda)$; $\mathsf{h} = \perp$; $\mathsf{reg} := \perp$; $\mathsf{gsk_h} = \perp$; $\Sigma := \perp$
- $(\mathsf{mpk}; \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{param})$
- $(\mathsf{mpk}, \mathsf{opk}; \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{param})$
- $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk})$
- $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$
- $(\mathsf{info}^*, \mathsf{reg}^*, m^*, \pi^*) \leftarrow \mathcal{A}^{\mathsf{SndToHU},\mathsf{SignHU}}(\mathsf{st}_\mathcal{A})$
- If $\mathbf{Judge}(\mathsf{gpk}, \mathsf{info}^*, \mathsf{reg}^*, m^*, \Sigma, \pi^*) = 0$ return 0
- If $\mathsf{reg} \neq \mathsf{reg}^*$ return 1, else return 0

Fig. 12. Opening soundness game. Dashed text is omitted in the single authority setting, whereas in the separate authorities setting Dashed text replaces the preceding line.

3. Static and Partially Dynamic Group Signatures

In the previous section, we defined fully dynamic group signatures. Earlier formal definitions covered static groups, where the membership is fixed at initialization [14], and partially dynamic groups where new members may enroll but without revocation [19,43]. We will now discuss how our definitions for fully dynamic group signatures can be relaxed to the static and partially dynamic settings, and we will show that aside from minor differences earlier formal definitions of group signatures can be seen as restrictions of our definitions to special cases.

3.1. *Restriction to Partially Dynamic Signatures*

In prior works, the term dynamic group signature refers to the case where new members may enroll at any time, but members cannot leave the group once they have joined. This partially dynamic setting is simply a special case of our fully dynamic group signatures, where we never revoke members, i.e., in all calls to Update we have revocation set $\mathcal{R} = \emptyset$.

For some designs of group signatures, the group information does not change as new members are enrolled. This is unlike the fully dynamic setting, where by necessity the group information must change to prevent revoked signers from using their current keys to produce valid signatures. When the group information is immutable, we can eliminate the **UpdateGroup** function entirely from our scheme and remove the corresponding Update oracle in the security definitions. This in turn means we only have one epoch $\tau = 0$, and also the **IsActive** policy now simply says that an enrolled user is active immediately after the group manager considers her joining procedure to have ended successfully. Since info_0 is generated by GM together with the manager public key mpk, we can without loss of generality assume $\mathsf{info}_0 = \varepsilon$. For the special case of partially dynamic group signatures with immutable group information, the definitions can therefore be simplified by excluding the epoch $\tau = 0$, the public group information $\mathsf{info}_0 = \varepsilon$ and the **UpdateGroup** function.

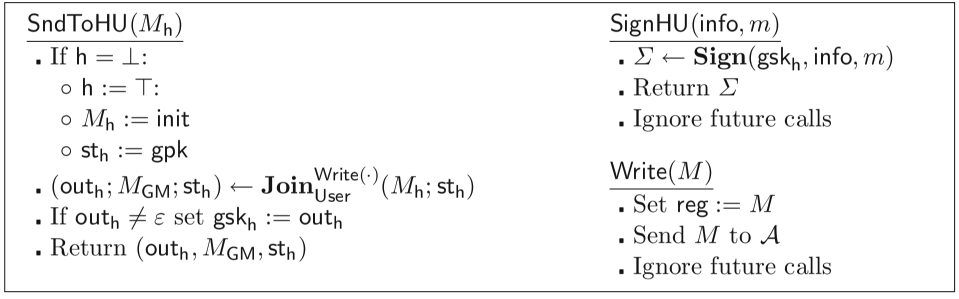

Fig. 13. Oracles used in the opening soundness game.

These notational simplifications lead us to the following syntax for a partially dynamic group signature scheme with immutable group information:

Reg: A data structure with records reg_i for joining session identifiers $\mathsf{i} \in \mathbb{N}$ is associated with the following oracles:

- $\mathsf{ReadReg}(\mathsf{i})$: Returns reg_i (or $\perp$ if no such record exists).
- $\mathsf{WriteReg}(\mathsf{i}, M)$: Sets $\mathsf{reg}_\mathsf{i} := M$ and ignores further calls with the same i.

GSetup$(1^\lambda) \to \mathsf{param}$: A PPT algorithm generating trusted parameters (or 1^λ if there is no trusted setup).
GKGen$(\mathsf{param}) \to (\mathsf{mpk}; \mathsf{st}_\mathsf{GM})$: A PPT algorithm for group manager key generation. The group public key is $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk})$.
If we separate the roles of group manager GM and opening authority OA, they instead run the interactive key generation protocol

$$((\mathsf{mpk}; \mathsf{st}_\mathsf{GM}); (\mathsf{opk}, \mathsf{osk})) \leftarrow \langle \mathbf{GKGen}_\mathsf{GM}(\mathsf{param}); \mathbf{GKGen}_\mathsf{OA}(\mathsf{param}) \rangle$$

and let $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$ be the group public key.
Join: An interactive protocol for enrolling a user in the group. It is defined by the following PPT algorithms:

- $\mathbf{Join}_\mathsf{User}^{\mathsf{WriteReg}(\mathsf{i},\cdot)}(M; \mathsf{st}) \to (\mathsf{out}; M_\mathsf{GM}; \mathsf{st})$.
- $\mathbf{Join}_\mathsf{GM}^{\mathsf{ReadReg}(\mathsf{i})}(\mathsf{i}, M_\mathsf{GM}; \mathsf{st}_\mathsf{GM}) \to (\mathsf{out}_\mathsf{GM}; M; \mathsf{st}_\mathsf{GM})$.

IsActive$(\mathsf{i}, \mathsf{st}_\mathsf{GM}) \to 1/0$: A DPT algorithm defining when a user is considered active. In the partially dynamic case, the policy is that the user joining in session i is active if and only if the group manager terminated with success symbol $\top$.
Sign$(\mathsf{gsk}, m) \to \Sigma$: A PPT signing algorithm.
Verify$(\mathsf{gpk}, m, \Sigma) \to 1/0$: A DPT verification algorithm.
Open$(\mathsf{gpk}, \mathsf{st}_\mathsf{GM}, m, \Sigma) \to (\mathsf{i}, \pi)$: A PPT opening algorithm.
If we separate the roles of group manager and opening authority, OA instead uses the opening key osk to run $\mathbf{Open}^{\mathsf{ReadReg}(\cdot)}(\mathsf{gpk}, \mathsf{osk}, m, \Sigma) \to (\mathsf{i}, \pi)$.
Judge$(\mathsf{gpk}, \mathsf{reg}, m, \Sigma, \pi) \to 1/0$: A DPT algorithm determining if signature was correctly attributed to registry record reg.

The corresponding simplified security experiments for partially dynamic group signature scheme with immutable group information are given in Fig. 14. We list the experiments for the single group manager setting where we omit the dashed text. The dashed text replaces the preceding line in the separate authority setting. The oracles are defined exactly as in the case of fully dynamic group signatures and simplified by excluding the group information $\mathsf{info} = \varepsilon$ and epochs $\tau = 0$.

Comparison to Bellare et al. [19] Model. Bellare et al. [19] define partially dynamic group signatures with separate group manager (called Issuer) and opening authority (called Opener). Their definition is a specific type of partially dynamic group signature with immutable group information. We will now describe restrictions to our definition of partially dynamic group signatures that yields a definition similar to their definition.

Bellare, Shi and Zhang consider a group manager state of the form $\mathsf{st}_\mathsf{GM} = (\mathsf{msk}, \{\mathsf{st}_\mathsf{GM}^i\})$. The state is therefore compartmentalized to consist of a fixed part msk,

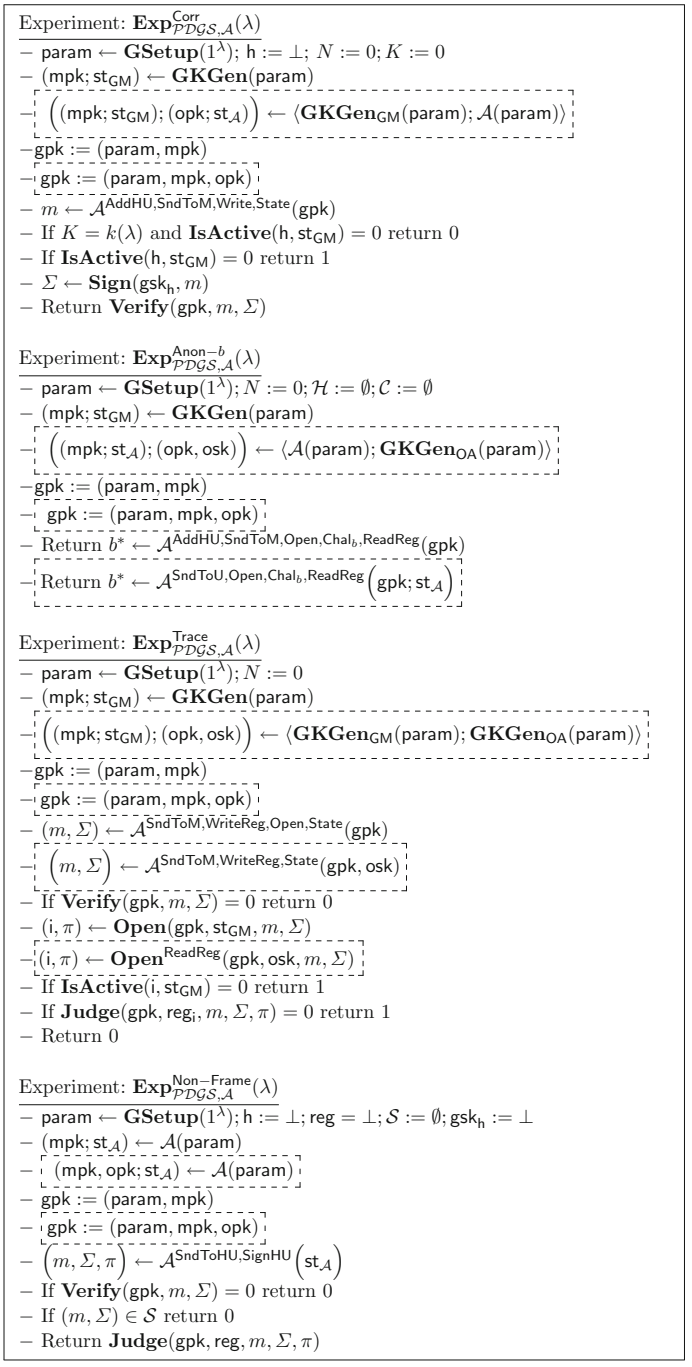

Fig. 14. Security experiments for partially dynamic group signatures. Dashed text is omitted in the single authority setting, whereas it replaces the preceding line in the separate authorities setting.

which we call the manager's secret key, and other parts $\mathsf{st}^{\mathsf{i}}_{\mathsf{GM}}$ that are specific to the joining sessions. It is assumed joining session states are independent of each other, which makes it easier to reason about concurrent joins. In particular, since the joins are independent of each other, we can under the same assumption of compartmentalized group manager state define correctness in terms of a single joining session and ignore any concurrent joining sessions.

Bellare, Shi and Zhang assume a trusted key generation procedure. If we assume keys to be honestly generated, we can combine the trusted parameter generation and the key generation protocol into a single algorithm, which first runs the parameter generation and then honestly executes the interactive key generation protocol. So there is little loss of generality in assuming a single trusted algorithm that generates all keys.

Taken together, for a partially dynamic group signature scheme with compartmentalized group manager state and trusted key generation we can simplify the syntax and security experiments as described below. A group signature scheme with compartmentalized group manager state satisfying our definition in Fig. 14 directly yields a partially dynamic group signature scheme satisfying the definition in Fig. 15 by letting the key generation procedure run the setup algorithm and the interactive key generation protocol honestly and outputting the resulting keys.

Reg : A data structure with records $\mathsf{reg}_{\mathsf{i}}$ for joining session identifiers $\mathsf{i} = 1, 2, 3$, etc., which is associated with the following oracles:

- $\mathsf{ReadReg}(\mathsf{i})$: Returns $\mathsf{reg}_{\mathsf{i}}$ (or $\perp$ if no such record exists).
- $\mathsf{WriteReg}(\mathsf{i}, M)$: Sets $\mathsf{reg}_{\mathsf{i}} := M$ and ignores further calls with the same i.

GKGen$(1^{\lambda}) \to (\mathsf{gpk}, \mathsf{msk}, \mathsf{osk})$: A trusted PPT algorithm for key generation.
Join : An interactive protocol for enrolling a user in the group and is defined by the following PPT algorithms:

- $\mathbf{Join}^{\mathsf{WriteReg}(\mathsf{i},\cdot)}_{\mathsf{User}}(M; \mathsf{st}) \to (\mathsf{out}; M_{\mathsf{GM}}; \mathsf{st})$.
- $\mathbf{Join}^{\mathsf{ReadReg}(\mathsf{i})}_{\mathsf{GM}}(\mathsf{i}, M_{\mathsf{GM}}, \mathsf{msk}; \mathsf{st}^{\mathsf{i}}_{\mathsf{GM}}) \to (\mathsf{out}_{\mathsf{GM}}; M; \mathsf{st}^{\mathsf{i}}_{\mathsf{GM}})$.

IsActive$(\mathsf{i}, \mathsf{st}_{\mathsf{GM}}) \to 1/0$: A DPT algorithm defining when a user is considered active. The policy is that the user joining in session i is active if and only if the group manager terminated with success symbol $\top$.
Sign$(\mathsf{gsk}, m) \to \Sigma$: A PPT signing algorithm.
Verify$(\mathsf{gpk}, m, \Sigma) \to 1/0$: A DPT verification algorithm.
$\mathbf{Open}^{\mathsf{ReadReg}(\cdot)}(\mathsf{gpk}, \mathsf{osk}, m, \Sigma) \to (\mathsf{i}, \pi)$: A PPT opening algorithm.
Judge$(\mathsf{gpk}, \mathsf{reg}, m, \Sigma, \pi) \to 1/0$: A DPT algorithm determining if signature was correctly attributed to registry record reg.

The matching simplified security experiments are given in Fig. 15. The oracles are defined exactly as in the previous definitions.

Bellare, Shi and Zhang assume a confidential communication channel between the group manager and joining users, while we assume an open channel. A scheme satisfying our security definition will of course also satisfy the weaker security definition that assumes confidential communication channels. Conversely, the group manager and user can use public-key cryptography to establish a confidential channel, so this definitional difference is immaterial. Bellare, Shi and Zhang also consider a slightly stronger

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{BSZ},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $\mathsf{h} := \bot$; $N := 0; K := 0$
– $m \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{Write}}(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk})$
– If $K = k(\lambda)$ and $\mathbf{IsActive}(\mathsf{h}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 0
– If $\mathbf{IsActive}(\mathsf{h}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1
– $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{h}}, m)$
– Return $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma)$

Experiment: $\mathbf{Exp}^{\mathsf{Anon}-b}_{\mathcal{BSZ},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $N := 0; \mathcal{H} := \emptyset; \mathcal{C} := \emptyset$
– Return $b^* \leftarrow \mathcal{A}^{\mathsf{SndToU},\mathsf{Open},\mathsf{Chal}_b,\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{msk})$

Experiment: $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{BSZ},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $N := 0$
– $\left(m, \Sigma\right) \leftarrow \mathcal{A}^{\mathsf{SndToM},\mathsf{WriteReg}}(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk})$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– $(\mathsf{i}, \pi) \leftarrow \mathbf{Open}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{osk}, m, \Sigma)$
– If $\mathbf{IsActive}(\mathsf{i}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1
– If $\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}_{\mathsf{i}}, m, \Sigma, \pi) = 0$ return 1
– Return 0

Experiment: $\mathbf{Exp}^{\mathsf{Non}-\mathsf{Frame}}_{\mathcal{BSZ},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $\mathsf{h} := \bot; \mathsf{reg} = \bot; \mathcal{S} := \emptyset; \mathsf{gsk}_{\mathsf{h}} := \bot$
– $\left(m, \Sigma, \pi\right) \leftarrow \mathcal{A}^{\mathsf{SndToHU},\mathsf{SignHU}}(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk})$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– If $(m, \Sigma) \in \mathcal{S}$ return 0
– Return $\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}, m, \Sigma, \pi)$

Fig. 15. Security experiments for partially dynamic group signatures akin to Bellare et al. [19].

definition of correctness where an honestly generated signature must always open to identify the signer, which as discussed in Sect. 2.3 is covered in a computational sense by traceability and therefore in our opinion immaterial.

It can now be seen by direct comparison with [19] that the experiments in Fig. 15 yield a security definition very similar to the one given by Bellare, Shi and Zhang. One remaining difference is that Bellare, Shi and Zhang explicitly include a public-key infrastructure where each user has an identity $j \in \mathbb{N}$ and generates a key pair $(\mathsf{upk}[j], \mathsf{usk}[j])$, and they consider the registry to be under the jurisdiction of the group manager. However, as discussed in Sect. 2.2 we can eliminate this extra step to simplify definitions and just consider the registry record to be under control of the user. For concreteness, this can be done by letting the user generate a key pair $(\mathsf{upk}, \mathsf{usk})$ for a digital signature scheme that is strongly existentially unforgeable under chosen message attack (sEUF-CMA secure) and create a signature σ on her intended record together with her identity j and public key $\mathsf{upk}[j]$, i.e., in Bellare et al. let the record created when a user is joining in session i

be $\mathsf{reg}[\mathsf{i}] = (j, \mathsf{upk}[j], \mathsf{reg}_\mathsf{i}, \sigma)$. The user can now send this record to the group manager when she wants to write to the registry. Since the group manager cannot forge the user's signature, this is equivalent to letting the user have one-time write access to the registry, and given the record index i, it is easy to map to the user identity j. Our definition in Fig. 15 is mostly similar to the definition of Bellare et al. [19] modulo this difference.

Comparison to Kiayias and Yung [43]. Kiayias and Yung [43] also give a formal definition of partially dynamic group signatures with immutable group information. Their definition is in the single authority setting and deviates from our definition and Bellare et al. [19] by trusting the group manager to open honestly and not requiring proofs of correct attribution to a signer. This simplification is easy to implement given a single authority partially dynamic group signature; the opening algorithm can simply discard the proof of correct attribution. Kiayias and Yung assume the key generation process is honest and they also assume the group manager's internal state can be compartmentalized as $\mathsf{st}_\mathsf{GM} = (\mathsf{msk}, \{\mathsf{st}^\mathsf{i}_\mathsf{GM}\})$, both of which are special cases of our definition.[5] In their model, opening takes place against a public record, so we let the opening algorithm have access to the registry, while it is of course easy to just incorporate it into the state of the group manager. We present the syntax and security experiments below.

Reg : A data structure with records reg_i for joining session identifiers $\mathsf{i} = 1, 2, 3$, etc., which is associated with the following oracles:

- $\mathsf{ReadReg}(\mathsf{i})$: Returns reg_i (or $\perp$ if no such record exists).
- $\mathsf{WriteReg}(\mathsf{i}, M)$: Sets $\mathsf{reg}_\mathsf{i} := M$ and ignores further calls with the same i.

$\mathbf{GKGen}(1^\lambda) \to (\mathsf{gpk}; \mathsf{msk})$: A PPT algorithm for group manager key generation.
Join : An interactive protocol for enrolling a user in the group. It is defined by the following PPT algorithms:

- $\mathbf{Join}^{\mathsf{WriteReg}(\mathsf{i},\cdot)}_{\mathsf{User}}(M; \mathsf{st}) \to (\mathsf{out}; M_\mathsf{GM}; \mathsf{st})$.
- $\mathbf{Join}^{\mathsf{ReadReg}(\mathsf{i})}_{\mathsf{GM}}(\mathsf{i}, M_\mathsf{GM}, \mathsf{msk}; \mathsf{st}^\mathsf{i}_\mathsf{GM}) \to (\mathsf{out}_\mathsf{GM}; M; \mathsf{st}^\mathsf{i}_\mathsf{GM})$.

$\mathbf{IsActive}(\mathsf{i}, \mathsf{st}_\mathsf{GM}) \to 1/0$: A DPT algorithm defining when a user is considered active. The policy is that the user joining in session i is active if and only if the group manager terminated with success symbol $\top$.
$\mathbf{Sign}(\mathsf{gsk}, m) \to \Sigma$: A PPT signing algorithm.
$\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) \to 1/0$: A DPT verification algorithm.
$\mathbf{Open}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{msk}, m, \Sigma) \to \mathsf{i}$: A DPT opening algorithm.

The matching security experiments are given in Fig. 16. The oracles are defined exactly as in the previous definitions.

The security definition given in Fig. 16 is close to the security definition given by Kiayias and Yung. There are some immaterial differences, such as allowing arbitrary identifier i versus numbering them consecutively, and their specifying that the registry must have entries of a specific form corresponding to the joining transcript. There is one significant difference in the anonymity experiment. Here Kiayias and Yung require indistinguishability of two signers as long as the keys are consistent with the protocol, i.e., could plausibly have been generated. We on the other hand, just require indistin-

[5] Kiayias and Yung use very different notation. What they call "state" is what we call "registry."

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{KY},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $\mathsf{h} := \bot$; $N := 0$; $K := 0$
– $m \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{SndToM},\mathsf{WriteReg},\mathsf{State}}(\mathsf{gpk})$
– If $K = k(\lambda)$ and $\mathbf{IsActive}(\mathsf{h}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 0
– If $\mathbf{IsActive}(\mathsf{h}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1
– $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{h}}, m)$
– Return $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma)$

Experiment: $\mathbf{Exp}^{\mathsf{Anon}-b}_{\mathcal{KY},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $N := 0; \mathcal{H} := \emptyset; \mathcal{C} := \emptyset$
– Return $b^* \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{SndToM},\mathsf{Open},\mathsf{Chal}_b,\mathsf{ReadReg}}(\mathsf{gpk})$

Experiment: $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{KY},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}) \leftarrow \mathbf{GKGen}(1^\lambda)$
– $(m, \Sigma) \leftarrow \mathcal{A}^{\mathsf{SndToM},\mathsf{WriteReg},\mathsf{Open},\mathsf{State}}(\mathsf{gpk})$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– $\mathsf{i} \leftarrow \mathbf{Open}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{msk}, m, \Sigma)$
– If $\mathbf{IsActive}(\mathsf{i}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1, else return 0

Experiment: $\mathbf{Exp}^{\mathsf{Non}-\mathsf{Frame}}_{\mathcal{KY},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $\mathsf{h} := \bot; \mathcal{S} := \emptyset$
– $\left(m, \Sigma\right) \leftarrow \mathcal{A}^{\mathsf{SndToHU},\mathsf{SignHU}}(\mathsf{gpk}, \mathsf{msk})$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– If $(m, \Sigma) \in \mathcal{S}$ return 0
– If $\mathbf{Open}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{msk}, m, \Sigma) = \mathsf{h}$ return 1, else return 0

Fig. 16. Security experiments for partially dynamic group signatures akin to Kiayias and Yung [43].

guishability for honestly generated keys. In our opinion, the stronger anonymity notion of Kiayias and Yung is overkill; it is reasonable to assume honest signers will follow the protocol, and therefore, our anonymity experiment suffices to protect them. Aside from this difference, inspection reveals that there are mainly notational and terminological differences between our security definition in Fig. 16 and Kiayias and Yung [43].

3.2. *Restriction to Static Group Signatures*

In static group signatures, the set of group members is fixed from the start and never changes. This means the setup procedure is different, since it includes the key generation for all the group members, so strictly speaking static group signatures are not just a definitional restriction of dynamic group signatures but a distinct definition altogether. However, we can easily convert a dynamic group signature into a static one by incorporating the join procedure for the users into the key generation process and then at the end hand them their secret keys.

In our definition of static group signatures, we will replace the group manager key generation with a trusted combined key generation protocol that also generates keys for

the group members. Since the set of group members is static, there are no managerial operations taking place so the sole purpose of the group manager is to be able to open group signatures and identify the signer. We therefore only consider the single authority setting. Also, since membership does not change, the group information does not need to change either and we only have a single epoch, so we can without loss of generality fix $\mathsf{info}_0 = \varepsilon$ and $\tau = 0$ and omit them from the definition. We can also eliminate reference to the **IsActive** procedure since all group members are automatically considered active. Finally, since enrollment of users takes place during setup, we can only get non-frameability if the registry is honestly generated. This means that in all the security experiments, we must assume trusted key generation, and therefore, we can incorporate the trusted parameters generation into the key generation procedure.

With these changes in mind, we get the following security experiments for static group signatures, where N is an arbitrary polynomially bounded function of the security parameter.

Reg: A data structure with records reg_i for members numbered $\mathsf{i} = 1, 2, 3, \ldots, N$. It is associated with the following algorithms:

- $\mathsf{ReadReg}(\mathsf{i})$: Returns reg_i (or $\perp$ if no such record exists).
- $\mathsf{WriteReg}(\mathsf{i}, M)$: Sets $\mathsf{reg}_\mathsf{i} := M$ and ignores further calls with the same i.

$\mathbf{GKGen}^{\mathsf{WriteReg}}(1^\lambda, N) \to (\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_\mathsf{GM})$: A PPT algorithm for group manager key generation, which depends on the number of desired group members N. Returns the group public key, secret keys for the users, writes registry entries $\mathsf{reg}_1, \ldots, \mathsf{reg}_N$, and sets the state of the group manager (which can be interpreted as a group manager secret key).
$\mathbf{Sign}(\mathsf{gsk}, m) \to \Sigma$: A PPT signing algorithm.
$\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) \to 1/0$: A DPT verification algorithm.
$\mathbf{Open}(\mathsf{gpk}, \mathsf{st}_\mathsf{GM}, m, \Sigma) \to (\mathsf{i}, \pi)$: A PPT opening algorithm.
$\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}, m, \Sigma, \pi) \to 1/0$: A DPT algorithm determining if a signature was correctly attributed to registry record reg.

The matching simplified security experiments for static group signature schemes and oracle Keys are given in Fig. 17. The rest of the oracles are defined exactly as in the case of fully dynamic group signatures and simplified by excluding the group information $\mathsf{info} = \varepsilon$ and epochs $\tau = 0$.

Comparison to static group signature definitions in [14]. Bellare et al. [14] gave a formal definition of static group signatures where group membership is fixed at the beginning of the protocol. Their definition does not include proof of correct opening; they just require the group manager to identify the signer. This can be seen as a special case of our static group signature scheme, where we omit the **Judge** algorithm and simply trust the judgment of the group manager. We still need the core properties of traceability, i.e., we can open all valid signatures to one of the members, and still need partial non-frameability, in that a signature cannot be opened to a member who did not sign it. We present the simplified definition of static group signatures in Fig. 18 and the syntax below. Oracles are defined as for the previous definitions. Bellare et al. [14] combine the traceability and non-frameability notions into a combined notion they call

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{SGS},\mathcal{A},N}(\lambda)$
- $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}^{\mathsf{WriteReg}}(1^\lambda, N)$
- $(\mathsf{h}, m) \leftarrow \mathcal{A}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N, \mathsf{st}_{\mathsf{GM}})$
- If $\mathsf{h} \notin [N]$ return 1
- $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk}_\mathsf{h}, m)$
- Return $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma)$

Experiment: $\mathbf{Exp}^{\mathsf{Anon}-b}_{\mathcal{SGS},\mathcal{A},N}(\lambda)$
- $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}^{\mathsf{WriteReg}}(1^\lambda, N)$
- Return $b^* \leftarrow \mathcal{A}^{\mathsf{Open},\mathsf{Chal}_b,\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N)$

Experiment: $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{SGS},\mathcal{A},N}(\lambda)$
- $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}^{\mathsf{WriteReg}}(1^\lambda, N)$
- $(m, \Sigma) \leftarrow \mathcal{A}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N, \mathsf{st}_{\mathsf{GM}})$
- If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
- $(\mathsf{i}, \pi) \leftarrow \mathbf{Open}(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, m, \Sigma)$
- If $\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}_\mathsf{i}, m, \Sigma, \pi) = 0$ return 1
- Return 0

Experiment: $\mathbf{Exp}^{\mathsf{Non-Frame}}_{\mathcal{SGS},\mathcal{A},N}(\lambda)$
- $\mathcal{S} = \emptyset, \mathsf{h} = \perp$
- $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}^{\mathsf{WriteReg}}(1^\lambda, N)$
- $\left(m, \Sigma, \pi\right) \leftarrow \mathcal{A}^{\mathsf{ReadReg},\mathsf{Keys},\mathsf{SignHU}}\left(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}\right)$
- If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
- If $(m, \Sigma) \in \mathcal{S}$ return 0
- Return $\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}_\mathsf{h}, m, \Sigma, \pi)$

Oracle: $\mathsf{Keys}(\mathsf{h})$
- If $\mathsf{h} \notin [N]$ return $\perp$
- Return $\{\mathsf{gsk}_i\}_{i \neq \mathsf{h}}$ and ignore future calls

Fig. 17. Security experiments for static group signatures.

full traceability. It is not hard to see though that the definition given in Fig. 18 is almost equivalent to their security definition for static group signatures.

> $\mathbf{GKGen}(1^\lambda, N) \rightarrow (\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}})$: A PPT algorithm for group manager key generation, which depends on the number of desired group members N. Returns the group public key, secret keys for the users, and sets the state of the group manager (which can be interpreted as a group manager secret key).
> $\mathbf{Sign}(\mathsf{gsk}, m) \rightarrow \Sigma$: A PPT signing algorithm
> $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) \rightarrow 1/0$: A DPT verification algorithm.
> $\mathbf{Open}(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, m, \Sigma) \rightarrow \mathsf{i}$: A DPT opening algorithm.

If we have a static group signature scheme with proofs of correct opening satisfying the definition in Fig. 17, then it is easy to convert it into a similar group signature without proofs of correct opening satisfying the definition in Fig. 18. The group key generation

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{BMW},\mathcal{A},N}(\lambda)$
– $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(1^\lambda, N)$
– $(\mathsf{h}, m) \leftarrow \mathcal{A}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N, \mathsf{st}_{\mathsf{GM}})$
– If $\mathsf{h} \notin [N]$ return 1
– $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{h}}, m)$
– Return $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma)$

Experiment: $\mathbf{Exp}^{\mathsf{Anon}-b}_{\mathcal{BMW},\mathcal{A},N}(\lambda)$
– $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(1^\lambda, N)$
– Return $b^* \leftarrow \mathcal{A}^{\mathsf{Open},\mathsf{Chal}_b}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N)$

Experiment: $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{BMW},\mathcal{A},N}(\lambda)$
– $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(1^\lambda, N)$
– $(m, \Sigma) \leftarrow \mathcal{A}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N, \mathsf{st}_{\mathsf{GM}})$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– $\mathsf{i} \leftarrow \mathbf{Open}(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, m, \Sigma)$
– If $\mathsf{i} \notin [N]$ return 1
– Return 0

Experiment: $\mathbf{Exp}^{\mathsf{Non}-\mathsf{Frame}}_{\mathcal{BMW},\mathcal{A},N}(\lambda)$
– $\mathcal{S} = \emptyset, \mathsf{h} = \perp$
– $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(1^\lambda, N)$
– $\left(m, \Sigma\right) \leftarrow \mathcal{A}^{\mathsf{Keys},\mathsf{SignHU}}\left(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}\right)$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– If $(m, \Sigma) \in \mathcal{S}$ return 0
– If $\mathbf{Open}(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, m, \Sigma) = \mathsf{h}$ return 1, else return 0

Fig. 18. Security experiments for static group signatures akin to Bellare et al. [14].

algorithm can include registry record reg_i in the secret signing key gsk_i, and the group manager state $\mathsf{st}_{\mathsf{GM}}$ may include the entire registry (we could in principle instead include the registry information in the group public key, but this might lead to an undesirable increase in the size of gpk). When the **Open** algorithm is called, it runs the original opening algorithm to get (i, π), verifies the proof using the **Judge** algorithm, and returns i. It follows from the security definitions in Fig. 17 that this simple modification leads to a static group signature scheme without proof of correct opening that satisfies the definitions in Fig. 18.

4. On the Security of Some Existing Schemes

In this section, we look at existing constructions of fully dynamic group signatures and discuss their security in our model. Different design paradigms can use the group information in different ways. For example, the manager can use the group information to store the list of the active members, include an accumulator, or provide lists of the

users that have been revoked. The way the group information is used can also affect the timespan users are considered active, i.e., the set of epochs in which users are entitled to sign. In our model, this is configured by specifying the **IsActive** policy, which spells out the conditions governing the activation of a user. We can thus compare different constructions based on their **IsActive** policy.

In [7], Bootle et al. gave a generic construction of group signatures from accountable ring signatures. We start by showing that their generic construction is secure in the stronger variant of our model, namely with respect to separate authorities and adversarial key generation. We then look at some constructions based on revocation lists [48,49,53]. These implicitly use a weaker **IsActive** policy than allowed by the model and thus achieve a slightly weaker notion of traceability. While the small definitional gap may not necessarily constitute an issue for the applications, we show that small changes to the constructions allow for stronger policies.

4.1. *Bootle et al. Scheme [7]*

Bootle et al. [7] gave a generic construction of accountable ring signatures, where every signature can be traced back to a user in the ring. Differently from group signatures, accountable ring signatures lack appointed authorities. Instead signers choose their designated opening authority at the time of signing. Bootle et al. [7] outlined how to obtain fully dynamic group signatures (with a single authority) from accountable ring signatures. In addition, they gave an efficient instantiation in the random oracle model based on the DDH assumption. Their instantiation yields signatures of logarithmic size (in the size of the ring), while signing is quasi-linear, and signature verification requires a linear number of operations. Bootle et al. [7] instantiation is in some settings more efficient than existing group signature schemes based on standard assumptions.

In their generic construction, each user has a secret key and an associated verification key. To sign, users first encrypt their verification key. Then, via a membership proof, they provide a signature of knowledge showing that the verification key belongs to the ring, and that they know the corresponding secret key. We now reproduce their definitions and prove their construction is secure in our separate authorities model. Toward this end, we take the liberty of fleshing out some group management specific details that [9] did not fully specify, e.g., a two-party joining protocol. Other designs are also possible, and we will discuss the key points after the security proof.

Accountable Ring Signatures. Bootle et al. [7] define an accountable ring signature scheme over a PPT setup ARS_{Setup} as a tuple of polynomial time algorithms $(\mathsf{ARS}_{OKGen}, \mathsf{ARS}_{UKGen}, \mathsf{ARS}_{Sign}, \mathsf{ARS}_{Vfy}, \mathsf{ARS}_{Open}, \mathsf{ARS}_{Judge})$.

$\mathsf{ARS}_{Setup}(1^\lambda)$: Given the security parameter, produces public parameters pp used (sometimes implicitly) by the rest of the scheme. The public parameters define key spaces PK, DK, VK, SK which are for openers' and users' keys, respectively, with efficient algorithms for sampling and deciding membership.

$\mathsf{ARS}_{OKGen}(pp)$: Given the public parameters pp, produces a public key $pk \in PK$ and secret key $dk \in DK$ for an opener. Without loss of generality, we assume dk defines pk deterministically and write $pk = \mathsf{ARS}_{OKGen}(pp, dk)$ when computing pk from dk.

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathsf{ARS},\mathcal{A}}(\lambda)$
- $pp \leftarrow \mathsf{ARS}_{Setup}(1^\lambda)$
- $(vk, sk) \leftarrow \mathsf{ARS}_{UKGen}(pp)$
- $(pk, R, m) \leftarrow \mathcal{A}(pp, sk)$
- $\sigma \leftarrow \mathsf{ARS}_{Sign}(pk, sk, R, m)$
- If $pk \notin PK$ or $R \not\subset VK$ or $vk \notin R$ return 1
- If $\mathsf{ARS}_{Vfy}(pk, R, m, \sigma) = 1$ return 1
- Return 0

Fig. 19. Correctness game.

$\mathsf{ARS}_{UKGen}(pp)$: Given the public parameters pp, produces a verification key $vk \in VK$ and a secret signing key $sk \in SK$ for a user. We can assume sk deterministically determines vk and write $vk = \mathsf{ARS}_{UKGen}(pp, sk)$ when computing vk from sk.

$\mathsf{ARS}_{Sign}(pk, sk, R, m)$: Given an opener's public key, a user's secret key, a ring (i.e. a set of verification keys), and a message, produces a ring signature σ. The algorithm returns the error symbol $\perp$ if $pk \notin PK, R \not\subset VK, sk \notin SK$ or $vk = \mathsf{ARS}_{UKGen}(pp, sk) \notin R$.

$\mathsf{ARS}_{Vfy}(pk, R, m, \sigma)$: Given an opener's public key, a ring, a message, and a signature, returns 1 if accepting the signature and 0 otherwise. We assume the algorithm always returns 0 if $pk \notin PK$ or $R \not\subset VK$.

$\mathsf{ARS}_{Open}(dk, R, m, \sigma)$: Given an opener's secret key, a ring, a message, and a ring signature, returns a verification key vk and a proof ψ that the owner of vk produced the signature. If $dk \notin DK$ or σ is not a valid signature using $pk = \mathsf{ARS}_{OKGen}(pp, dk)$, the algorithm returns $\perp$.

$\mathsf{ARS}_{Judge}(pk, R, vk, m, \sigma, \psi)$: Given an opener's public key, a ring, a verification key, a message, a ring signature, and an opening proof, returns 1 if accepting the proof and 0 otherwise. We assume the algorithm returns 0 if σ is invalid or $vk \notin R$.

Accountable ring signatures should be correct, anonymous, traceable, fully unforgeable, and tracing sound. We recall [7] definitions of all these properties as follows (Fig. 19).

Definition 8. (*Correctness*) An accountable ring signature scheme is *correct* if for all PPT adversaries $\mathcal{A}$,

$$\Pr[\mathbf{Exp}^{\mathsf{Corr}}_{\mathsf{ARS},\mathcal{A}}(\lambda) = 1] \approx 1.$$

Anonymity ensures that a signature does not reveal the identity of the ring member who produced it without the opener explicitly wanting to open the particular signature. The definition below implies anonymity against full key exposure attacks [13] as in the game the adversary is allowed to choose the secret signing keys of the users (Figs. 20, 21).

Experiment: $\mathbf{Exp}^{\mathsf{Anon}}_{\mathsf{ARS},\mathcal{A}}(\lambda)$
- $Q_{\text{Sign}} := \emptyset$
- $pp \leftarrow \mathsf{ARS}_{Setup}(1^\lambda)$
- $b \leftarrow \{0,1\}$
- $(pk, dk) \leftarrow \mathsf{ARS}_{OKGen}(pp)$
- $b^* \leftarrow \mathcal{A}^{\text{Chal}_b,\text{Open}}(pp, pk)$
- If $b = b^*$ return 1
- Return 0

Fig. 20. Anonymity game.

$\text{Chal}_b(R, m, sk_0, sk_1)$
- $\sigma_0 \leftarrow \mathsf{ARS}_{Sign}(pk, sk_0, R, m)$
- $\sigma_1 \leftarrow \mathsf{ARS}_{Sign}(pk, sk_1, R, m)$
- $Q_{\text{Sign}} := ((R, m, \sigma_0), (R, m, \sigma_1))$
- If $\sigma_0 = \perp$ or $\sigma_1 = \perp$ return $\perp$
- Return σ_b

$\text{Open}(R, m, \sigma)$
- If $(R, m, \sigma) \in Q_{\text{Sign}}$ return $\perp$
- Return $\mathsf{ARS}_{Open}(dk, R, m, \sigma)$

Fig. 21. Oracles used in the anonymity game.

Definition 9. (*Anonymity*) An ARS scheme is *anonymous* if for all PPT adversaries $\mathcal{A}$,

$$\Pr[\mathbf{Exp}^{\mathsf{Anon}}_{\mathsf{ARS},\mathcal{A}}(\lambda) = 1] \approx \frac{1}{2}$$

- Chal_b: is an oracle that the adversary can only call once. On query (R, m, sk_0, sk_1), it produces signatures on m for ring R under both keys and, if successful, returns σ_b, otherwise it returns $\perp$.
- Open: is an oracle that on a query (R, m, σ) returns $\mathsf{ARS}_{Open}(dk, R, m, \sigma)$. If σ was obtained by calling Chal_b on $(R, m, \cdot, \cdot)$, the oracle returns $\perp$.

Traceability ensures that the specified opener can always identify the ring member who produced a signature and that she is able to produce a valid proof for her decision (Fig. 22).

Definition 10. (*Traceability*) An accountable ring signature scheme is *traceable* if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible

$$\mathbf{Adv}^{\mathsf{Trace}}_{\mathsf{ARS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{Trace}}_{\mathsf{ARS},\mathcal{A}}(\lambda) = 1].$$

Full unforgeability ensures that an adversary, who may control the opener, can neither falsely accuse an honest user of producing a ring signature nor forge ring signatures on behalf of an honest ring. The former should hold even when all other users in the ring are corrupt (Figs. 23, 24).

Experiment: $\mathbf{Exp}^{\mathsf{Trace}}_{\mathsf{ARS},\mathcal{A}}(\lambda)$
- $pp \leftarrow \mathsf{ARS}_{Setup}(1^\lambda)$
- $(dk, R, m, \sigma) \leftarrow \mathcal{A}(pp)$
- $pk \leftarrow \mathsf{ARS}_{OKGen}(pp, dk)$
- $(vk, \psi) \leftarrow \mathsf{ARS}_{Open}(dk, R, m, \sigma)$
- If $\mathsf{ARS}_{Vfy}(pk, R, m, \sigma) = 0$ return 0
- If $\mathsf{ARS}_{Judge}(pk, R, vk, m, \sigma, \psi) = 1$ return 0
- Return 1

Fig. 22. Traceability game.

Experiment: $\mathbf{Exp}^{\mathsf{Unforge}}_{\mathsf{ARS},\mathcal{A}}(\lambda)$
- $Q_{\mathrm{UKGen}} := \emptyset; S_{\mathrm{UKGen}}[\cdot] := \perp; Q_{\mathrm{RevealU}} := \emptyset; Q_{\mathrm{Sign}} := \emptyset$
- $pp \leftarrow \mathsf{ARS}_{Setup}(1^\lambda)$
- $(pk, vk, R, m, \sigma, \psi) \leftarrow \mathcal{A}^{\mathrm{UKGen,Sign,RevealU}}(pp)$
- If $(pk, vk, R, m, \sigma) \in Q_{\mathrm{Sign}}$ return 0
- If $vk \in Q_{\mathrm{UKGen}} \setminus Q_{\mathrm{RevealU}}$ and $\mathsf{ARS}_{Judge}(pk, R, vk, m, \sigma, \psi) = 1$ return 1
- If $R \not\subseteq Q_{\mathrm{UKGen}} \setminus Q_{\mathrm{RevealU}}$ or $(pk, \cdot, R, m, \sigma) \in Q_{\mathrm{Sign}}$ return 0
- If $\mathsf{ARS}_{Vfy}(pk, R, m, \sigma) = 1$ return 1
- Return 0

Fig. 23. Full unforgeability game.

Definition 11. (*Full Unforgeability*) An ARS scheme is *fully unforgeable* if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible

$$\mathbf{Adv}^{\mathsf{Unforge}}_{\mathsf{ARS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{Unforge}}_{\mathsf{ARS},\mathcal{A}}(\lambda) = 1].$$

- UKGen: runs $(vk, sk) \leftarrow \mathsf{ARS}_{UKGen}(pp)$ and returns vk. Q_{UKGen} is the set of verification keys vk that have been generated by this oracle.
- Sign: is an oracle that on query (pk, vk, R, m) checks if $vk \in R \cap Q_{\mathrm{UKGen}}$, in which case returns a signature σ on m under the key corresponding to vk and ring R. Q_{Sign} contains the queries and responses.
- RevealU: is an oracle that when queried on $vk \in Q_{\mathrm{UKGen}}$ returns the corresponding signing key sk. Q_{RevealU} is the list of verification keys vk for which the corresponding signing key has been revealed.

Tracing soundness is analogous to our opening binding definition for group signatures and ensures that a signature cannot trace to two different users in the ring. Namely, only one person can be identified as the signer even when all users as well as the opener are fully corrupt (Fig. 25).

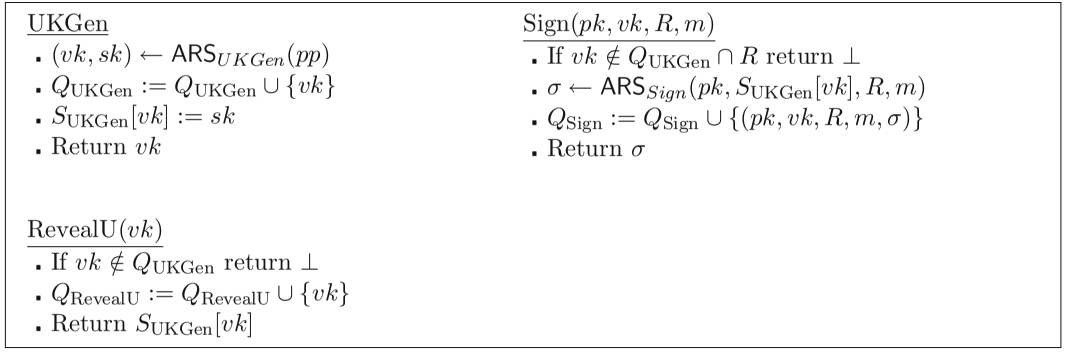

Fig. 24. Oracles used in the full unforgeability game.

Experiment: $\mathbf{Exp}^{\mathsf{Trace-sound}}_{\mathsf{ARS},\mathcal{A}}(\lambda)$
- $pp \leftarrow \mathsf{ARS}_{Setup}(1^\lambda)$
- $(pk, R, m, \sigma, vk_1, \psi_1, vk_2, \psi_2) \leftarrow \mathcal{A}(pp)$
- If $vk_1 = vk_2$ return 0
- If $\mathsf{ARS}_{Judge}(pk, R, vk_1, m, \sigma, \psi_1) = 0$ return 0
- If $\mathsf{ARS}_{Judge}(pk, R, vk_2, m, \sigma, \psi_2) = 0$ return 0
- Return 1.

Fig. 25. Traceability game.

Definition 12. (*Tracing Soundness*) An ARS scheme is *traceable* if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible

$$\mathbf{Adv}^{\mathsf{Trace-sound}}_{\mathsf{ARS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{Trace-sound}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1].$$

Security in Our Model. Next we show that the [7] construction of a fully dynamic group signature scheme obtained from an accountable ring signature is secure in our security model with separate authorities. We note that [7] does not fully specify the construction, e.g., it does not specify the registration protocol and only consider a single authority. Therefore, the construction described in Fig. 26 is not unique and alternatives are possible.

A user initiates a joining session with the group manager by generating a key pair (vk, sk), using ARS_{UKGen}, and then adds the verification key vk into the registry. The manager reads the registry entry and checks if the same key had already been registered, in which case the joining fails. We do not spell out the details of the registry, but we assume users can write once into the registry and that the manager can read the content of it. We recall that a registry offering these features can be instantiated with a PKI and thus we abstract it out to simplify the construction.

The group manager stores the outcome of all the joining sessions as well as the lists $\mathcal{I}_\tau$ of active users at each epoch. To activate new members, the manager keeps a list of users that have joined in the current epoch and updates the group information info, which triggers a new epoch. The information of the group info consists of the verification keys of all active members. A group signature consist of an accountable ring signature using the current information info as the ring and the opening authority public key as the opener's key. Our construction of a fully dynamic group signature from an accountable ring signature is presented in Fig. 26.

Theorem 1. *The generic group signature scheme construction from accountable ring signatures of Fig. 26 satisfies our separate authority definitions for a secure, fully dynamic group signature scheme, and is additionally opening binding.*

Proof. We will use a similar proof strategy for all properties: we will assume the existence of an adversary $\mathcal{A}$ against the corresponding property of the group signature

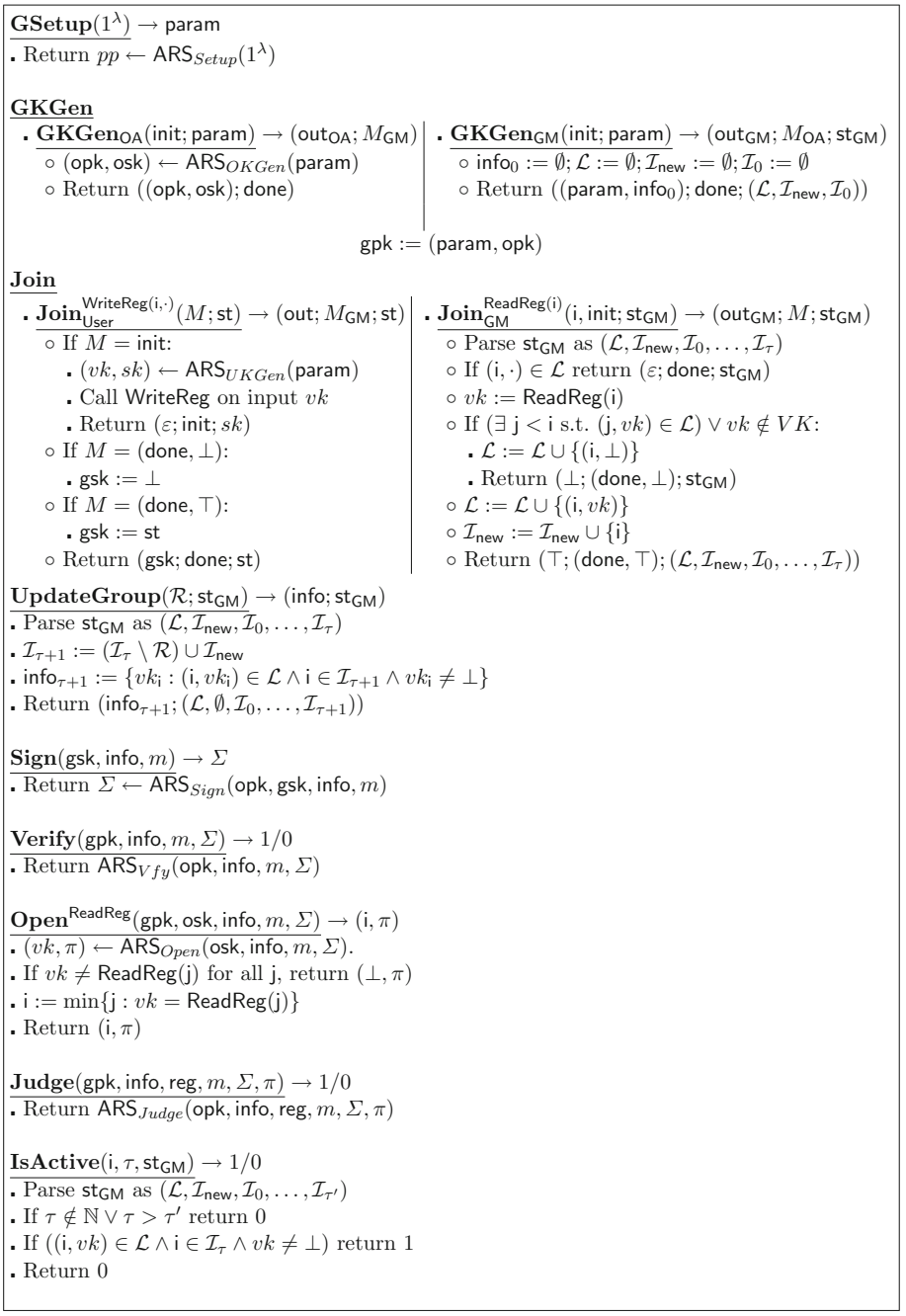

Fig. 26. Construction of a fully dynamic group signature from an accountable ring signature [7].

scheme. We will then show how to build an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the same property of the accountable ring signature.

For correctness, let $\mathcal{B}$ be the adversary in the correctness experiment of Definition 8. The adversary receives pp and the secret key sk of the target user. Given pp the adversary, $\mathcal{B}$ provides param to $\mathcal{A}$ and let her pick the public key opk for the opening authority of the group signature, while $\mathcal{B}$ plays the role of the honest group manager in the game of Fig. 1. The adversary $\mathcal{B}$ generates the initial group information info_0 and provides it to $\mathcal{A}$. When simulating AddHU for $\mathcal{A}$, $\mathcal{B}$ uses the secret key sk for the challenge user; note that vk can be efficiently obtained given sk. The oracle calls to SndToM, Update, Write, State used by $\mathcal{A}$ are also simulatable by $\mathcal{B}$ which keeps the internal state of the group manager. Once $\mathcal{A}$ returns a message and epoch pair (m, τ), $\mathcal{B}$ retrieves the group information info_τ, consisting of the public keys of active group members at epoch τ, and returns $(\mathsf{opk}, m, \mathsf{info}_\tau)$. We observe that $\mathcal{A}$ only wins the game of Fig. 1 when either the registration protocol fails to complete successfully for the target user, or the target user is flagged as inactive even though she has joined and is not revoked, or if the produced signature fails to verify. The registration of the target user fails only in case $\mathcal{A}$ already successfully registered the same verification key vk in a previous session. Since vk is not exposed to $\mathcal{A}$ before AddHU is called, this corresponds to guessing the target key vk. Assuming the key space VK is large enough this only happens with small probability and we can thus assume the target user to successfully complete the joining protocol. When $\mathcal{B}$ updates the group information, the target user verification key is included in info until explicitly removed by another update. The verification key can only be removed either by revoking the target user or if the adversary adds another group member with the same vk and then requests to revoke her. However, registering the same verification key in a later session of the joining protocol would cause the new session to fail. The remaining winning condition of $\mathcal{A}$ in the game of Fig. 1 is captured by the last line of Definition 8.

For anonymity, we follow the same strategy: the adversary $\mathcal{B}$ plays the game of Definition 9 and obtains the public parameters pp and a public key pk. Then he proceeds to simulate the key generation protocol for the group signature (Fig. 5): he lets $\mathcal{A}$ to generate the group manager public key and sets pk as the opening public key of the opening authority. Note that the key generation protocol in Fig. 26 consists of each authority independently producing and announcing their own keys. This implies that pk is sufficient for $\mathcal{B}$ to simulate the protocol. Finally, $\mathcal{B}$ needs to simulate $\mathcal{A}$'s oracle calls. For the challenge and opening oracles, this is done by using his own oracles, whereas for SndToU, $\mathsf{ReadReg}$ it can simply answer directly. We note that $\mathcal{B}$ will correctly guess the challenge if and only if $\mathcal{A}$ is correct as well.

For traceability, $\mathcal{B}$ starts by receiving pp and internally running both sides of the key generation protocol. He then starts the group signature traceability game of Fig. 7 and calls $\mathcal{A}$. As $\mathcal{B}$ plays the role of the group manager in $\mathcal{A}$'s game, he can directly reply to her oracle queries. To complete the proof, we examine when $\mathcal{A}$ wins her game: it must be the case that user i is inactive or that the **Judge** algorithm fails. In the accountable ring signature definition, the first case folds into the second: i being inactive implies $vk_i \notin R$ which will explicitly cause the judging algorithm to fail. In either case, $\mathcal{A}$ succeeding in the game of Fig. 7 implies $\mathcal{B}$ succeeding according to Definition 10.

For non-frameability, let $\mathcal{B}$ play in the full unforgeability game of Definition 11. The adversary $\mathcal{B}$ receives pp and initializes $\mathcal{A}$ in the game of Fig. 9, which generates the group public key gpk. $\mathcal{B}$ uses his oracles UKGen and Sign to respond to the queries of the $\mathsf{SndToHU}$ and SignHU, respectively. Adversary $\mathcal{B}$ forwards $\mathcal{A}$'s output together with the opener public key and the honest user verification key. A wining output by $\mathcal{A}$ in her game will cause $\mathcal{B}$ to win via the first branch on his own game: note that $\mathcal{B}$ does not make use of his RevealU oracle and that $\mathcal{A}$ does not output a user identity since she only includes a single honest user in the group.

Reduction for opening binding is near trivial from tracing soundness: $\mathcal{B}$ simply passes the setup parameters and converts the output of $\mathcal{A}$ into a ring. We complete the proof by pointing out that under the accountable ring signatures definitions ARS_{Judge} implies correct verification. □

Alternative Constructions. The accountable ring signature of [7,8] describes how one can construct dynamic group signatures from accountable ring signatures, but it omits the group management details. To amend this, we specified a simple joining protocol, but other options are possible. Furthermore, our protocol assumes the existence of an ideal register which could be instantiated in different ways. We do point out, however, that for the traceability proof to go through, it is necessary that the manager should not accept keys that have already being registered. At the same time, it is important that the registry needs to be robust enough, such that attempts to modifying or copying entries will fail. The former is captured by our idealization of the registry, which we recall can be realized with a PKI. The latter is accomplished by considering valid only the first occurrence of a key in the registry, and ignoring all later occurrences as the group manager should have rejected the corresponding joining sessions.

4.2. *Constructions Based on Revocation Lists*

A common approach for designing efficient fully dynamic group signatures is by using revocation lists. Users interact with the group manager to obtain a certificate for their group membership. The group information, periodically updated by the manager, consists of a revocation list which stores information about revoked members. To sign, users have to prove they hold a valid certificate and that they are not part of the set of revoked users. Examples of efficient schemes following this approach include the ones by Libert et al. [48,49] and Nakanishi et al. [53].

In these constructions, a user is considered authorized to sign until the manager explicitly includes her in the revocation list. However, this means that a user can also sign with respect to old epochs, including those predating her joining to the group. As the user was not yet part of the group at that time, it raises the question to whether this represents an issue for the security of the scheme.

At first glance, this is the dual of a well-known issue with many revocation systems. If a user is revoked and anonymity is maintained, the revoked user is able to produce back-dated signatures that still verify. The difference here is that while the revoked user *was* authorized to be part of the group for the epoch in question, in the situation described above, however, the signing user was in fact *not* part of the group at that time. If the adversary is able to block the opening of this signature (e.g., via legal action), its

existence would implicitly frame the group's past membership as the signature would be attributed to them.

The issue resides mainly on the interpretation one gives to the epochs in the lifespan of the scheme. Namely, whether epochs reference the state of the group at the time the corresponding group information was created. Our model does not opt for a specific choice, as different design paradigms may have different takes on this. To capture different options, our model includes the **IsActive** algorithm for spelling out the conditions that makes a user active. This helps to clear potential ambiguities a construction may have regarding the timespan with respect to which users are allowed to sign. Moreover, it also enables to compare the security achieved by different schemes based on the strictness of their underlying **IsActive** policy.

The **IsActive** policy has to satisfy some necessary requirements, imposed by our model, which ensure the definitions capture the intended security notions. One of these requirements is that if user i is associated with a joining session where the group manager ended her part successfully *before* info_τ was created, and user i is not revoked at or before epoch τ, the algorithm returns 1. However, the policy does not impose a specific outcome in case the joining session terminates *after* the info_τ was created, which could then be set equal to either 0 or 1. For example, we can include the following condition into the policy:

- If i is associated with a joining session where the group manager completed her part *after* info_τ was created, the algorithm returns 0.

Observe that this requirement is achieved by the policy of the construction in Fig. 26. On the other hand, this policy may be too strict for constructions following the revocation list approach, as members can typically sign with respect to epochs predating their enrollment into the group. The violation of the above condition translates into a trivial attack against traceability: an adversary can simply enroll a user and then return a signature by this user with respect to an epoch predating her joining epoch. If the signature is valid, this may represent a breach of traceability because the user is not regarded as active with respect to that epoch. We notice that in case such a policy was adopted, constructions such as [48,49,53] would be all susceptible to this attack. For such constructions we can then replace the above condition with the following one, which was implicitly assumed in their respective models:

- If i is associated with a joining session where the group manager completed her part *successfully* after info_τ was created, and user i is not revoked at or before epoch τ, the algorithm returns 1.

Note that the **IsActive** algorithm receives as input the internal state of the group manager; thus, the outcome of the policy can depend on whether the joining session for user i has currently terminated. This enables to capture the instant activation of certificate based constructions, i.e., users become active as soon as they successfully terminate the join protocol, without requiring further action from the group manager.

Given the above trivial attack, it seems that [48,49,53] can only achieve a slightly weaker notion of traceability than the construction given in Fig. 26. Whether the difference on the two notions is substantial may depend on the intended applications of the primitive. In the following, we briefly recall the [48,49,53] constructions and sug-

gest some simple modifications to prevent the above trivial attack, which allow them to achieve traceability with respect to a stronger policy.

Libert et al. Scheme [48]. In [48], users are assigned leaves of a complete binary tree and given a membership certificate containing a unique tag identifying the user, and a commitment to the path from the root to the user's leaf in the tree. Note that the certificate is not bound to the epoch at which the user joined the group. In fact, users joining does not change info_τ or the epoch τ itself.

Revocation is based on the subset difference method [56], using disjoint sets S_{k_i,u_i} for $i = 1, \ldots, m$ which cover non-revoked users. Sets are represented by two nodes, a node k_i and one of its descendants node u_i, and cover all leaves of the sub-tree rooted at node k_i which are not leaves of the sub-tree rooted at u_i. Revocations trigger a new epoch and the update of info_τ with a new cover.

To sign, the group member anonymously proves that she holds a membership certificate and that the node indicated by the certificate belongs to one of those sets. More precisely, the user proves that her leaf is a descendant of node k_i but not a descendant of node u_i for some $i \in [m]$.

Since user certificates are not bound to epochs and leaves are covered until their corresponding users are revoked, members are able to produce valid signatures with respect to revocation lists preexisting their joining of the group. Therefore, the construction is susceptible to the previous trivial attack against traceability. A similar argument also applies to the variant of the scheme given in [49].

A possible countermeasure against this is to regard unassigned leaves as revoked until they are assigned. In this way, the revocation list is used to store the current state of the group at a certain epoch. This is simple to do as the scheme bounds the maximum number of users. We do, however, need to re-examine the number of subsets in the revocation list required to express this, as the $2|\mathcal{R}| - 1$ bound for $|\mathcal{R}|$ revoked users may now seem impractical. If we assume leaves are allocated sequentially to users, we can bound the number of subsets by $2|\mathcal{R}_1| + \log(|\mathcal{N} \setminus \mathcal{R}_2|)$ where $\mathcal{R}_2$ is the set of leaves pending allocation, $\mathcal{R}_1$ is the set of leaves allocated to users who were later revoked, and $\mathcal{N}$ the set of all leaves. Thus, our fix is only marginally more expensive than the base system and much more efficient than a naive analysis would indicate.

Nakanishi et al. Scheme [53]. The scheme of Nakanishi et al. [53] is another certificate-based scheme in the random oracle model. It achieves constant time for both signing and signature verification, relative to the size of the group and the number of revoked users.

A user's group membership certificate consists of a signature on (x, ID) produced by the group manager, where x is a secret owned by the user and ID is a unique integer the manager assigned to her. The group manager can revoke users by issuing revocation lists info_τ. Each list consists of a sequence of *open* integer intervals (R_i, R_{i+1}) signed by the manager, whose endpoints are all the revoked ID's. At each epoch τ, a signer fetches the current info_τ and proves, as part of the signature, that her ID is contained in one interval of the revocation list. If the ID lies between two revoked users' identities, it means it is not an endpoint and so she has not been revoked.

Again, verifiers only know of revoked members, not active ones and, similarly to [48], the time of joining is not taken into account. This allows users to sign with respect to any epoch prior to them joining the group.

The scheme could be easily immunized against the trivial attack against traceability. A first solution, as for [48], is to initialize the revocation list with all ID's of users that have not joined the group yet. When the manager assigns an ID to a new user, she updates reg and the revocation list info_τ. This way, the signature size is not affected. On the other hand, revocation lists are now proportional to the size of the maximum number of users, instead of the number of revoked users.

An alternative countermeasure requires the group manager to include the joining epochs in the certificates by signing $(x, \mathrm{ID}, \tau_{\mathrm{join}})$, where x is a secret owned by user ID and τ_{join} is the joining epoch. A signer then needs to include in the signature a proof that τ_{join} is not greater than the signing epoch. To realize the latter, one can use membership proof techniques from [22,60] which are already used in the original scheme for proving that the ID lies in an interval (R_i, R_{i+1}). This would increase the cost of signing and verifying by only a constant factor. The new membership proof would require the group manager to provide signatures for every elapsed epoch, which could then be included in info_τ along with the revocation list.

References

[1] G. Ateniese, J. Camenisch, S. Hohenberger, B. de Medeiros, Practical group signatures without random oracles. *IACR Cryptology ePrint Archive* (2005)

[2] G. Ateniese, J. Camenisch, M. Joye, G. Tsudik, A practical and provably secure coalition-resistant group signature scheme, in *Advances in Cryptology - CRYPTO* (2000)

[3] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, M. Ohkubo, Structure-preserving signatures and commitments to group elements. *J. Cryptology*. **29**(2):363–421, (2016)

[4] M. Abdalla, B. Warinschi, On the minimal assumptions of group signature schemes, in *ICICS*, vol. 3269 of *Lecture Notes in Computer Science* (2004)

[5] D. Boneh, X. Boyen, H. Shacham, Short group signatures, in *Advances in Cryptology - CRYPTO* (2004)

[6] E.F. Brickell, J. Camenisch, L. Chen, Direct anonymous attestation, in *Conference on Computer and Communications Security - CCS* (2004)

[7] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, J. Groth, C. Petit, Short accountable ring signatures based on DDH, in *Computer Security - ESORICS* (2015)

[8] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, J. Groth, Foundations of fully dynamic group signatures, in *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings* (2016)

[9] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, J. Groth, Foundations of fully dynamic group signatures. *IACR Cryptology ePrint Archive* (2016)

[10] P. Bichsel, J. Camenisch, G. Neven, N.P. Smart, B. Warinschi, Get shorty via group signatures without encryption, in *Security and Cryptography for Networks - SCN* (2010)
[11] O. Blazy, D. Derler, D. Slamanig, R. Spreitzer, Non-interactive plaintext (in-)equality proofs and group signatures with verifiable controllable linkability, in *Topics in Cryptology - CT-RSA* (2016)
[12] M. Backes, L. Hanzlik, J. Schneider, Membership privacy for fully dynamic group signatures. *IACR Cryptology ePrint Archive*. **2018**:641, (2018)
[13] A. Bender, J. Katz, R. Morselli, Ring signatures: Stronger definitions, and constructions without random oracles. *Journal of Cryptology*. **22**(1):114–138, (2009)
[14] M. Bellare, D. Micciancio, B. Warinschi, Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions, in *Advances in Cryptology - EUROCRYPT* (2003)
[15] M. Bellare, P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols, in *Conference on Computer and Communications Security - CCS* (1993)
[16] E. Brickell, An efficient protocol for anonymously providing assurance of the container of a private key. *Submitted to the Trusted Computing Group* (2004)
[17] E. Bresson, J. Stern, Efficient revocation in group signatures, in *Public Key Cryptography - PKC* (2001)
[18] D. Boneh, H. Shacham, Group signatures with verifier-local revocation, in *Conference on Computer and Communications Security - CCS* (2004)
[19] M. Bellare, H. Shi, C. Zhang, Foundations of group signatures: The case of dynamic groups, in *Topics in Cryptology - CT-RSA* (2005)
[20] X. Boyen, B. Waters, Compact group signatures without random oracles, in *Advances in Cryptology - EUROCRYPT* (2006)
[21] X. Boyen, B. Waters, Full-domain subgroup hiding and constant-size group signatures, in *Public Key Cryptography - PKC* (2007)
[22] J. Camenisch, R. Chaabouni, A. Shelat, Efficient protocols for set membership and range proofs, in *Advances in Cryptology - ASIACRYPT* (2008)
[23] J. Camenisch, J. Groth, Group signatures: Better efficiency and new theoretical aspects, in *Security in Communication Networks - SCN* (2004)
[24] J. Camenisch, A. Lysyanskaya, Dynamic accumulators and application to efficient revocation of anonymous credentials, in *Advances in Cryptology - CRYPTO* (2002)
[25] J. Camenisch, A. Lysyanskaya, Signature schemes and anonymous credentials from bilinear maps, in *Advances in Cryptology - CRYPTO* (2004)
[26] J. Camenisch, M. Michels, A group signature scheme with improved efficiency, in *Advances in Cryptology - ASIACRYPT* (1998)
[27] D. Chaum, E. van Heyst, Group signatures, in *Advances in Cryptology - EUROCRYPT* (1991)
[28] Y. Dodis, A. Kiayias, A. Nicolosi, V. Shoup, Anonymous identification in ad hoc groups, in *Advances in Cryptology - EUROCRYPT* (2004)
[29] C. Delerablée, D. Pointcheval, Dynamic fully anonymous short group signatures, in *Progressing Cryptology - VIETCRYPT* (2006)
[30] D. Derler, D. Slamanig, Highly-efficient fully-anonymous dynamic group signatures, in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04–08, 2018* (2018)
[31] K. Emura, G. Hanaoka, Y. Kawai, T. Matsuda, K. Ohara, K. Omote, Y. Sakai, Group signatures with message-dependent opening: Formal definitions and constructions. *Security and Communication Networks* (2019)
[32] A. El Kaafarani, S. Katsumata, R. Solomon, *Anonymous reputation systems achieving full dynamicity from lattices*, in *Financial Cryptography and Data Security - 22st International Conference, FC 2018* (2018)
[33] J. Furukawa, H. Imai, An efficient group signature scheme from bilinear maps, in *Information Security and Privacy - ACISP* (2005)
[34] J. Furukawa, S. Yonezawa, Group signatures with separate and distributed authorities, in *Security in Communication Networks - SCN* (2004)
[35] E. Ghadafi, Efficient distributed tag-based encryption and its application to group signatures with efficient distributed traceability, in *Progress in Cryptology - LATINCRYPT* (2014)

[36] L. Garms, A. Lehmann, Group signatures with selective linkability, in *Public-Key Cryptography - PKC* (2019)
[37] J. Groth, Simulation-sound NIZK proofs for a practical language and constant size group signatures, in *Advances in Cryptology - ASIACRYPT* (2006)
[38] J. Groth, Fully anonymous group signatures without random oracles, in *Advances in Cryptology - ASIACRYPT* (2007)
[39] J.Y. Hwang, S. Lee, B.-H. Chung, H.S. Cho, D. Nyang, Short group signatures with controllable linkability, in *Lightweight Security & Privacy: Devices, Protocols and Applications (LightSec)* (2011)
[40] S. Krenn, K. Samelin, C. Striecks, Practical group-signatures with privacy-friendly openings, in *Availability, Reliability and Security - ARES* (2019)
[41] A. Kiayias, Y. Tsiounis, M. Yung, Traceable signatures, in *Advances in Cryptology - EUROCRYPT* (2004)
[42] A. Kiayias, M. Yung, Group signatures with efficient concurrent join, in *Advances in Cryptology - EUROCRYPT* (2005)
[43] A. Kiayias, M. Yung, Secure scalable group signature with dynamic joins and separable authorities. *IJSN*. **1**(1/2):24–45 (2006)
[44] B. Libert, S. Ling, F. Mouhartem, K. Nguyen, H. Wang, Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions, in *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4–8, 2016, Proceedings, Part II* (2016)
[45] A. Langlois, S. Ling, K. Nguyen, H. Wang, Lattice-based group signature scheme with verifier-local revocation, in *Public-Key Cryptography - PKC* (2014)
[46] B. Libert, S. Ling, K. Nguyen, H. Wang, Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors, in *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II* (2016)
[47] S. Ling, K. Nguyen, H. Wang, Y. Xu, *Lattice-based group signatures: Achieving full dynamicity (and deniability) with ease*. *CoRR*, arXiv:1801.08737 (2018)
[48] B. Libert, T. Peters, M. Yung, Group signatures with almost-for-free revocation, in *Advances in Cryptology - CRYPTO* (2012)
[49] B. Libert, T. Peters, M. Yung, Scalable group signatures with revocation, in *Advances in Cryptology - EUROCRYPT* (2012)
[50] B. Libert, D. Vergnaud, Group signatures with verifier-local revocation and backward unlinkability in the standard model, in *Cryptology and Network Security - CANS* (2009)
[51] R.W.F. Lai, T. Zhang, S.S.M. Chow, D. Schröder, Efficient sanitizable signatures without random oracles, in *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26–30, 2016, Proceedings, Part I* (2016)
[52] T. Nakanishi, N. Funabiki, Verifier-local revocation group signature schemes with backward unlinkability from bilinear maps, in *Advances in Cryptology - ASIACRYPT* (2005)
[53] T. Nakanishi, H. Fujii, Y. Hira, N. Funabiki, Revocable group signature schemes with constant costs for signing and verifying, in *Public Key Cryptography - PKC* (2009)
[54] T. Nakanishi, T. Fujiwara, H. Watanabe, A linkable group signature and its application to secret voting. *Transactions of Information Processing Society of Japan*. **40**(7):3085–3096 (1999)
[55] L. Nguyen, Accumulators from bilinear pairings and applications, in *Topics in Cryptology - CT-RSA* (2005)
[56] D. Naor, M. Naor, J. Lotspiech, Revocation and tracing schemes for stateless receivers, in *Advances in Cryptology - CRYPTO* (2001)
[57] L. Nguyen, R. Safavi-Naini, Efficient and provably secure trapdoor-free group signature schemes from bilinear pairings, in *Advances in Cryptology - ASIACRYPT* (2004)
[58] D.X. Song, Practical forward secure group signature schemes, in *Conference on Computer and Communications Security - CCS* (2001)
[59] Y. Sakai, J.C.N. Schuldt, K. Emura, G. Hanaoka, K. Ohta, On the security of dynamic group signatures: Preventing signature hijacking, in *Public Key Cryptography - PKC* (2012)
[60] I. Teranishi, K. Sako, k-times anonymous authentication with a constant proving cost, in *Public Key Cryptography - PKC* (2006)

[61] G. Tsudik, S. Xu, Accumulating composites and improved group signing, in *Advances in Cryptology - ASIACRYPT* (2003)