Mutual Accountability Layer: Accountable Anonymity within Accountable Trust

Vanesa Daza[1,3], Abida Haque*, Alessandra Scafuro*, Alexandros Zacharakis**, and Arantxa Zapico†

[1] Pompeu Fabra University
{vanesa.daza,alexandros.zacharakis, arantxa.zapico}@upf.edu
[2] North Carolina State University, Raleigh, USA
{ahaque3, ascafur}@ncsu.edu
[3] Cybercat

Abstract. Anonymous cryptographic primitives reduce the traces left by the users when interacting over a digital platform. However, they also prevent a platform owner to hold users accountable in case of malicious behaviour. Revocable anonymity offers a compromise by allowing only the manager (and not the other users) of the digital platform to de-anonymize user's activities when necessary. However, such de-anonymization power can be abused too, as a misbehaving manager can de-anonymize all the activities without user's awareness. Previous work propose to mitigate this issue by distributing the de-anonymization power across several entities. However, there is no comprehensive and formal treatment where both accountability and non-frameability (i.e., the inability to falsely accuse a party of misbehavior) for *both* the user and the manager are *explicitly* defined and provably achieved.

In this paper we formally define *mutual accountability*: a user can be held accountable for her otherwise anonymous digital actions and a manager is held accountable for *every* de-anonymization attempt; plus, no honest party can be framed – regardless of what malicious parties do.

Instead of distributing the de-anonymization power across entities, instead, we decouple the power of de-anonymization from the power of monitoring de-anonymization attempts. This allows for greater flexibility, particularly in the choice of the monitoring entities.

We show that our framework can be instantiated generically from threshold encryption schemes and succinct non-interactive zero-knowledge. We also show that the highly-efficient threshold group signature scheme by Camenisch et al.(SCN'20) can be modified and extended to instantiate our framework.

1 Introduction

We target the problem of *accountable* anonymity: an authorized user of a digital platform can generate a value anonymously[4], but when deemed necessary a value can be de-anonymized and linked to the identity of its creator. Accountability is naturally in tension with perfect anonymity because it is achieved by introducing some master trapdoor that enables a designated party to de-anonymize *any* message exchanged through the platform. Anonymity then holds conditionally on how the designated party *uses the trapdoor*.

Shouldn't the designated party be accountable for their de-anonymization activities? Shouldn't users at least be aware when de-anonymization activities take place?

Accountability: What is Missing from Previous Work. Previous works have recognized that the manager can abuse a master trapdoor [1,2,3,4,5,6]. The countermeasures proposed in the literature rely on distributing the role of the manager across n parties. That way, any de-anonymization can be performed only if a minimum number of parties agree (e.g., [7,8]). Other solutions [5,6] instead introduce a new party whose

* Research Supported by NSF grants #1012798,#1764025

** Research Supported by fellowships from "la Caixa" Foundation (ID 100010434). The fellowship codes are LCF/BQ/DI18/11660052 and LCF/BQ/DI18/11660053. Funding is also from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No. 713673.

[4] This work focuses on the *application* layer and not on the network layer. We assume that all users communicate over anonymous channels.

task is to permit the manager to de-anonymize the message. However, all existing definitions either lack generality or formality of the security goals for manager accountability, and consequently, their definitions and constructions can be applied only to specific scenarios. We elaborate on these issues below.

- Weak Manager-Accountability Guarantees. Most of the work (e.g., [1,8]) splits the manager into n parties. This approach lessens the manager's ability to abuse de-anonymization since multiple parties must agree to conduct the process. However, these works do not introduce any formal accountability guarantee against the managers, and often omit discussion on how the n parties are selected so that the underlying trust assumption can be satisfied.
- Lack of Comprehensive Definitions. Works such as [5,6] introduce a third entity (called "admitters" in [5]) that enables the manager to perform a de-anonymization. This approach introduces an additional layer of protection for the users, however, the formal security definition provided leaves out any protection for the manager. Specifically, it does not consider the non-frameability of the manager, making an implicit assumption on the good behavior of the admitters and users.
- Lack of Generality. Even in the more general framework where the de-anonymization power is controlled by two separate entities ([5]), the de-anonymization activity has a pre-defined granularity and is, therefore, suitable only in specific applications. For instance, in [5] the admitters provide a message-dependent trapdoor that allows a manager to de-anonymize *all messages* that have the same value (e.g., all messages that have the same date). This is useful for the application proposed in [5], where one wishes to de-anonymize all messages signed at a specific time with a single trapdoor, but not in general.
- Lack of Flexibility. In previous work, the manager accountability property is achieved by using parties who are also involved in the functioning of the platform (e.g., [7]), hence they might not be *independent of* the platform. Furthermore, there is an underlying assumption that these parties are fixed for the entire lifetime of the system. There is little discussion on how such parties are chosen and why they would be trustworthy and active throughout the lifetime of the system.

1.1 Our Contribution

In this work, we provide a general framework, the *mutual accountability layer*, to capture accountability guarantees for *both* the users and the managers. We believe that our definitions can enable the designing of mutually accountable systems for a wider set of applications than the ones considered so far in literature. We summarize our contributions as follows:

1. **A Formal Definition for Mutual Accountability.** We introduce the Mutual Accountability Layer (MAL), a cryptographic primitive that captures the properties of anonymity for the users, accountability and non-frameability for both the users and the manager. Our new definitions sharpen the ones provided in previous works, providing flexibility and verifiability properties. Accountability to all parties is guaranteed by introducing the **guardians**, a set of (potentially malicious) monitoring parties that oversees every de-anonymization process but it does not learn anything about the de-anonymized users. Security is guaranteed as long as the number of malicious guardians is below a threshold.
2. **Instantiations.** We provide a general template to construct a Mutual Accountability Layer that is based on a two-layer encryption: one for the guardians and one for the manager. Only if both collaborate, the identity of a user can be reconstructed. Besides our general construction, we also show how to modify the group signature scheme of [8] so that it can be used to instantiate a MAL scheme. The second approach is less general and is tied to the use of bilinear maps.
3. **Evolving Monitoring Parties.** We consider a dynamic setting where the monitoring parties, that is, the guardians, can change over time, and we discuss methods to *change the guard*. These are based on results on proactive secret sharing [9] and its more recent implementations [10].

We describe each contribution in the following sections.

1.2 Formal Definition for Mutual Accountability (MAL)

We initiate the formal treatment of *Mutual Accountability Layer* (MAL) where both the users and the manager are accountable. This layer can be added to any content that members of a digital platform generate, and as such, is not tied to any specific application. In contrast to previous work, we do not distribute the manager's role among n parties. We model two separate entities with different roles. The platform manager is the entity exclusively entitled to de-anonymize users, while the set of *guardians* are solely entitled to give permission to de-anonymize. The manager is unable to de-anonymize unless he generates a *publicly verifiable request of de-anonymization* for the guardians, declaring what value he wants to de-anonymize[5], and the guardians agree to collaborate. Even though the guardians must agree, they do not learn the identity of the misbehaving user. While the guardians could be composed of many parties, they are represented by a single public key. As we explain later, this allows for generality and applicability of this framework to the arbitrary designation of the guardians.

Threat Model and Security Guarantees. In our model, every party could act maliciously and collude. Specifically, the users, the manager, and **up to** t guardians can be fully malicious and can collude (for certain properties such as non-frameability, we even assume that all guardians are malicious). Within this threat model, we target the following security properties. **User anonymity:** Even if a malicious manager colludes with up to t malicious guardians, a message cannot be *covertly* de-anonymized (i.e., de-anonymized without a publicly verifiable request made by the manager). Moreover, even if all guardians are corrupted and collude, they still cannot de-anonymize any message when the manager is honest. **User non-frameability:** an honest user cannot not be falsely accused of being the creator of a value v that she did not generate. This should be true even if *all* the parties are malicious and collude (except the party who enrolls the users, though this can be avoided whenever the real identity can be proven cryptographically, we discuss this below). **User accountability:** a manager can de-anonymize any value if enough guardians cooperate. **Manager accountability:** it should be infeasible for a manager to de-anonymize a value v without leaving a publicly verifiable trace in the system. This property is guaranteed if at most t guardians are malicious. **Manager non-frameability:** even if users and guardians are malicious and collude, they should not be able to fabricate a de-anonymization request for which the manager will be held accountable.

Flexibility. Decoupling the manager role from the monitoring role is crucial for allowing great flexibility in the implementation of the system. First, in our definition, guardians are only involved in handling de-anonymization requests and take no role in the functioning of the system. They are "platform independent" and hence one could even use the same set of guardians for multiple platforms managed by distinct managers. This is in contrast with to previous proposals [11] where the parties performing the guardians' activity were also responsible for the functioning of the platform. Second, our definition identifies the set of guardians with a single public key and makes no assumption on the actual identity of the guardians. Specifically, the set of guardians can change over time – so long the same public key is maintained.

1.3 MAL: Instatiations

A General Instantiation of MAL We provide a general instantiation of MAL based on a threshold public-key encryption scheme (TES)[6] and a simulation-extractable non-interactive zero-knowledge proof system. Assume that a set of n guardians has been chosen (we describe selection mechanisms in Sec. 1.4) and assume that a threshold of them is honest (up to t could be arbitrarily malicious). First, the guardians will engage in a (non-interactive) protocol to compute a public key for the threshold encryption scheme $\mathsf{pk}_{\mathsf{TE}}^{\mathcal{GU}}$. Next, assume that the manager of the digital platform has published her public key $\mathsf{pk}^{\mathcal{GM}}$ for a CPA-secure encryption scheme. At high-level, a MAL is instantiated as follows. A user U_i becomes a member of the platform by enrolling with the key issuer using her "real identity", along with a new, freshly picked key vk_i that the user will use to be identified as a member of the platform. Here the meaning of "real identity" depends

[5] In some applications, the request should not be made public immediately.

[6] A threshold public-key encryption scheme is an encryption scheme where the secret key is split among n parties, and a ciphertext can be decrypted only if at least t shares of the secret keys are used.

on the application. For generality, we assume that there exists a procedure $\mathsf{Valid}(\mathrm{ID})$ that is applied to the real identity provided by the user. Once the ID is validated, the key issuer provides a signature σ_i on the pair $(\mathrm{ID}, \mathsf{vk}_i)$. The tuple $\mathsf{cert} = (\mathrm{ID}, \mathsf{vk}_i, \sigma_i)$ is then communicated to the manager of the platform. When the user generates a value v for the platform, she will send v along with an encryption of the identity $c_1 = \mathsf{Enc}_{\mathsf{pk}^{\mathcal{GM}}}(\mathsf{cert}_i)$ using the public key of the manager $\mathsf{pk}^{\mathcal{GM}}$ and a zero-knowledge proof of knowledge of the secret associated to vk_i and valid signatures σ_i computed by the key issuer. The size of the proof is *independent* of the number of authorized users enrolled in the digital service. To ensure *mutual* accountability, the ciphertext c_1 is wrapped inside another layer of encryption c_2, where c_2 was computed using the threshold encryption scheme under the public key of the guardians.

Thus, the final message posted by the user is $(\mathsf{m}, c_2, \mathsf{proof})$ where proof is a zero-knowledge proof that everything was computed correctly. We instantiate the TES with ElGamal Threshold Encryption Scheme [12,13], where the size of the public key is independent of the number of shares, n. Hence, the additional layer c_2 is succinct and independent of the number of guardians. If more than one set of guardians is available, the user can select the set of guardians that she trusts the most[7] and can add this to the tuple. By looking at the tuple $(\mathsf{m}, c_2, \mathsf{proof})$, the manager alone is unable to learn c_1 (and thus decrypt the identity vk_i), without having the guardians remove the layer of threshold decryption. This is true even if the manager colludes with up to t guardians. To de-anonymize a message, the manager must provide a publicly verifiable request for de-anonymization that at least $t+1$ guardians accept. We note that even if all guardians are fully malicious and decrypt every single instance, they still cannot de-anonymize any user without the secret key of the manager. While this approach is natural and is outlined in previous work, they did not provide formal definitions or proofs for accountability and non-frameability. We are the first to provide formal guarantees (the formal proof is available in the full version []). The scheme is described in Sec. 5

An Efficient Instantiation from Threshold Group Signature [8] Camenisch at al. [8] provide a practical t-out-of-n group signature scheme based on bilinear maps. Recall, in a group signature, a member of the group can sign anonymously within the group, but a group manager can de-anonymize any signature. In the scheme of Camenisch at al. [8], the manager is split in n parties, hence a signature can be de-anonymized if any subset of t managers agree. This scheme does not directly fit our setting, where we want only the manager to de-anonymize and the guardians should only allow this action. To fit our setting we modify their t-out-of-n scheme so that any subset of t guardians will only be able to remove one layer from the group signature. The other layer can be removed solely by the manager, and no other party will learn the decryption. More discussion is provided in Sec. 6.

1.4 Monitoring Committee: Selection and Evolution

The suitability of a selection procedure for guardians depends on the actual application. We outline some possibilities below.

(a) Selection among the users (only trust in your peers). When no external party is trusted, the guardians can be elected among the users enrolled in the platform using cryptographic sortition techniques. This can be implemented using Verifiable Random Function (VRF) [14] as follows. When a user registers in the system, she will choose a public key for a VRF. Then upon each "epoch" e, each user checks if she is elected as a *guardian* for the next epoch, by evaluation the VRF on input e, and check if the output is below a threshold ρ. This technique is used in Algorand [15] to select the committees that run the underlying consensus protocol. In our setting, we do not need a blockchain; we only need that the users of the digital platform have access to the public VRF keys of all users.

(b) Selection of external parties through voting mechanisms or by platform designers. When external parties that can be trusted exist, guardians could be selected through some voting mechanism among the platform designers. For instance, guardians can be chosen among nonprofit organizations that monitor citizen's rights in the US (such as the ACLU), etc. We stress that in our proof we do not need all the guardians to be honest. We tolerate up to t completely malicious guardians.

[7] For simplicity in this paper, we consider only one set of guardians.

(c) Selection through a public permissionless blockchain. Any blockchain that satisfies chain quality[8] can be used to select a committee of n guardians with the guarantee that at most t parties are malicious, where the parameters n, t are tied to the chain quality parameter [16,17,10]. The idea is the following: people who wish to be part of the guardians attempt to add a block to the blockchain containing a transaction with a public key that they want to use if they are selected. When enough blocks containing such transactions are stored on the blockchain, the public keys that appear in the first N blocks are automatically selected to be part of the guardians. *On Guardians' Incentives.* The incentive for external parties to participate in MAL comes from the application. For example, if guardians are chosen among nonprofit organizations their incentive for following the protocol follows from their social responsibility and reputation.

Evolving Committee. For more robustness of the system, the set of guardians changes periodically at epochs. To change the guardians, we propose to use a proactive secret-sharing mechanism for re-sharing the secret key sk among the new set of guardians using fresh shares which are independent than those of the old guardians. Proactive secret sharing techniques allow a secret to be*handed-off* between two sets of parties [9,18]. Specifically, we use the dynamic proactive secret sharing (DPSS) scheme of Goyal et al. [10]. In [10] the hand off works as follows: To hand off the shares of the secret, the old and new guardians first perform an initial computation that allows them to hold two independent sharings of the same random value r. The old committee can use the sharing of s and the sharing of r to reconstruct the value $s-r$ and publish it. Since r is random, the value $s-r$ leaks no information. Next, each member of the new committee adds $s-r$ to their own share of r. As a result, each member holds shares of $s-r+r=s$, the original secret. The new set of shares is independent of the old set of shares. This completes the hand-off of the secret from the old committee to the new committee. It is assumed that the old committee erases the old sharing after this phase is complete. Else, an adversary could slowly eventually corrupt the old committee afterwards and learn the secret. To end up with different sharings of r, each member C_i of the new committee picks a random value r_i and creates two different sharings of it. Then C_i shares ones of the shares with the old committee and the other with the new committee. Each party will then obtain a share for each r_i, it will sum their local share and hold a sharing of $r = r_1 + \ldots + r_n$, which is guaranteed to be random if one party provided a random r_i. To ensure that no party misbehaves, a polynomial commitment scheme is used to guarantee that all shares are well-formed. For details see Sec. 6.4.

2 Related Work

Several works [11,3,7] have explored the concept of accountable anonymity, but lack formal definitions of accountability and thus provable guarantees, and are suitable only to the communication layer. In such works, trusted mixers maintain the communication channel and are responsible for anonymization and de-anonymization.

More recently, Corrigan-Gibbs and Ford [19] targeted a closed group of people that self-manages the communications of its members and guarantees anonymity and some form of accountability. This work is specific for settings where the digital platform itself is decentralized, and it is not clear how one might extend it to other settings (e.g., where there is a platform manager).

Von Ahn et al. [1] note the threat of abuse of the de-anonymization power and proposes an anonymous and accountable system where the master secret key is not known by a single manager but is distributed (using some threshold scheme [20,21]) to a set of parties. This idea reduces the threat of abuse since the de-anonymization power is not concentrated in one entity. However, in [1], they provide only informal guarantees and do not discuss traceability or transparency of the de-anonymization process. Indeed, a subsequent work by Danezis and Sassaman [22] highlights that it could be arbitrary which messages get to remain anonymous and which ones will be censured. More importantly, it is unclear under what circumstances the parties are *provably* accountable for their de-anonymization activities.

More recently, Camenisch et al. [8] proposed dynamic group signatures; these are anonymous signatures that can be de-anonymized by a set of designated parties non-interactively (hence providing some form of

[8] Chain quality α_l means that in any sequence of l consecutive block at least α fraction of them are added by honest parties.

public traceability). While sufficient in certain scenarios, again this approach lacks the generality provided by our framework. Also, this line of work simply distributes the platform manager among several parties. We want manager accountability to be independent of the platform manager and to be enforced by a crowd.

Frankle et al. [2] discusses accountability within the context of electronic surveillance of platforms such as Facebook. Here the goal is to track secret law enforcement requests to digital platforms. This work is tailored to this specific setting where all parties (i.e., Facebook, FBI, judges) are assumed to act in good faith. In particular in this setting users have no anonymity to begin with, with respect to group manager (i.e., Facebook).

Libert and Joye [5]building on Sakai et al. [6], presented a group signature scheme with message-dependent opening. A dedicated committee, called the admitters, jointly decides if a message should be de-anonymized. If so, they jointly generate a *per-message* trapdoor that allows the manager to de-anonymize all instances that contain message m. As we mentioned earlier, there are important differences with our approach. First, in [5] there is no focus on the traceability guarantees of the de-anonymization procedure. Thus, users can still be de-anonymized unknowingly. Second, the de-anonymization is message-dependent instead of instance-dependent. Third, manager non-frameability is not considered, suggesting that admitters and manager are essentially the same authority, working towards the same goal, thus, admitters would not frame the manager. Like MAL, the signature has two layers of encryption, which must be removed by different entities. Message-dependent opening relies on *identity-based encryption* (IBE) scheme, specifically a fully collusion-resistant partially structure-preserving IBE, which is a variant of Waters' IBE scheme [23]. In IBE systems, a trusted party owns a master public and private key and private keys are signatures on the corresponding message [24]. A user asks for a private key from the key issuer, who can derive it from the master private key. Usually, the signature is on the user's identity [24], but here, the *messages* are the public key.

To sign, a user generates a two-level signature on their ID and the message they wish to sign m. Later, for de-anonymization, the guardians first remove the *message* layer and the authority removes the *identity* layer. The guardians generate a token t_m that depends on the message m. The fact that the admitters can generate a message dependent token inherently relies on using IBE. Upon receiving t_m, the authority can remove the other layer of encryption and then use t_m to decrypt the identity. The authority can then reuse the token t_m to de-anonymize *any* signatures on message m. Deviating from our work, the presence of a trusted key issuer is crucial to their system. [5] only achieve full traceability, while we have *non-frameability*. In full traceability, the adversary is *passive*, meaning she only *receives* keys. Meanwhile, non-frameability allows the adversary to *make* her own keys. This means it is important that the key issuer not be able to learn the private group keys of each user. Libert and Joye just assume that the key issuance is done honestly and that the private key is *erased* after all the members join. The adversary only gets secret keys off the admitter/guardians and opener/authority.

3 Preliminaries

Notation. We denote the set $\{1, 2, \ldots, n\}$ with $[n]$. We use $y \leftarrow \mathsf{F}(x)$ to indicate y is the output of a randomized algorithm F on input x and write $y \leftarrow \mathsf{F}(x; r)$ when we want to explicitly refer to the randomness r used. We use $y := \mathsf{F}(x)$ if F for a deterministic algorithm. PPT stands for probabilistic polynomial time. A function $\mathsf{negl}(n)$ is *negligible* if for every positive polynomial p there is an N such that for all integers $n > N$ it holds that $\mathsf{negl}(n) \leq \frac{1}{p(n)}$. We prove our protocols are secure with respect to PPT adversaries with a security parameter λ.

We use $(z, (y_i)_{U_i \in S}) \leftarrow \mathsf{F}\langle U_i(x_i)\rangle_{U_i \in S}(w)$ to denote a protocol between parties in a set S. Here, each party holds a secret input x_i and receives secret output y_i, z is the public output and w is the public input. In the context of our security games, to denote the execution of protocol F where an adversary, with auxiliary input u, participates in the execution and adaptively corrupts parties we write: $\left(z, \mathsf{st}_{\mathcal{A}}, (y_i)_{U_i \in H}\right) \leftarrow \mathsf{F}(\mathcal{A}(u))\langle U_i(x_i)\rangle_{U_i \in H}(w)$ where the parties in H are the honest parties, y_i denotes the output of honest party U_i, and $\mathsf{st}_{\mathcal{A}}$ denotes the output of the adversary $\mathcal{A}$.

Cryptographic Primitives. To instantiate our generic construction, we use standard cryptographic primitives such as a one-way function, a secure signature scheme, and a public key encryption scheme. Their syntax

and definitions can be found in any reference textbook (e.g., [25,26,27]). We also use non-interactive zero knowledge arguments of knowledge [28, Def. 2.5] which satisfy the stronger notion of simulation extractability [28, Def. 2.10]. Finally, we use a threshold encryption scheme which also satisfies a property of simulatable decryption.

For the DGS construction in 6, following [8], we use Pointcheval-Sanders signature scheme [29], various sigma protocols made non-interactive via the Fiat-Shamir transform [30], and a signature of knowledge [31].

We present the less standard definitions for simulation extractable nizk and threshold encryption with simulatable decryption in App. A.

4 Formal Definition of Mutual Accountability Layer

In this section, we describe MAL: we identify the parties in the protocol, describe the syntax of MAL in Definition 1, and finally describe the security properties.

- **Users**: The parties that generate messages. A user U can (1) join the platform with a valid identity ($\mathsf{JoinUser}^{\mathsf{Valid}}$) and obtain a public key for the platform; and (2) generate authorized value v in anonymously ($\mathsf{MemberAuth}$).
- **Key issuer:** This party, denoted by $\mathcal{KI}$, checks the identity of users and registers them (i.e., it helps execute $\mathsf{JoinUser}^{\mathsf{Valid}}$).
- **Manager:** This party, denoted by $\mathcal{GM}$, can request de-anonymization for a certain message ($\mathsf{ReqDeanon}$). If the request is granted, it learns the identity of the message creator (Deanon). The manager's requests are publicly verifiable.
- **Guardians:** The set of parties that grants access to a de-anonymization. These parties, denoted by $\{\mathsf{C}_1, \ldots, \mathsf{C}_n\}$, collectively protect the users against a potentially misbehaving group manager. They perform a one-time joint computation to compute a public key ($\mathsf{KeyGenGu}$). Then they monitor the de-anonymization requests generated by $\mathcal{GM}$. Once a request associated to v is validated, the guardians perform a joint computation to generate a value that will allow the group manager to trace the identity of the user who created v ($\mathsf{GrantDeanon}$). The outputs provided by the guardians is publicly verifiable.

Definition 1 (Mutual Accountability Layer Syntax). *A Mutually Accountable Layer* MAL *consists of the following PPT procedures:*

1. $\mathsf{pp} \leftarrow \mathsf{SetupParams}(1^\lambda)$. *On input security parameter λ, it outputs the parameters* pp *for the scheme. We assume* pp *implicitly contains the information about the message space $\mathcal{M}$, key space, etc.*
2. $(\mathsf{pk}^{\mathcal{KI}}, \mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}) \leftarrow \mathsf{KeyGenIssuer}(\mathsf{pp})$. *On input the parameters* pp *outputs a key pair $(\mathsf{pk}^{\mathcal{KI}}, \mathsf{sk}^{\mathcal{KI}})$ for $\mathcal{KI}$. It also initializes a state $\mathsf{st}^{\mathcal{KI}}$ used to maintain information of the members that join a group.*
3. $(\mathsf{pk}^{\mathcal{GM}}, \mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}}) \leftarrow \mathsf{KeyGenManager}(\mathsf{pp})$. *On input the parameters* pp*, it outputs a key pair for the group manager $(\mathsf{sk}^{\mathcal{GM}}, \mathsf{pk}^{\mathcal{GM}})$ and an initialized state $\mathsf{st}^{\mathcal{GM}}$ used to manage the group.*
4. $\left(\mathsf{pk}^{\mathcal{GU}}, (\mathsf{sk}_l^{\mathcal{GU}})_{\mathsf{C}_l \in \mathcal{GU}}\right) \leftarrow \mathsf{KeyGenGu}\,\langle \mathsf{C}_\ell(\cdot)\rangle_{\mathsf{C}_l \in \mathcal{GU}}(\mathsf{pp})$. *This is an interactive protocol between parties in a set $\mathcal{GU} := \mathsf{C}_1, \ldots, \mathsf{C}_n$. These parties, called guardians, have as a common public input the parameters* pp*. At the end of the protocol, each guardian receives as output a secret key $\mathsf{sk}_l^{\mathcal{GU}}$. All parties receive as output a public key $\mathsf{pk}^{\mathcal{GU}}$. We denote with* PK *the set public keys of the authorities, i.e.,* $\mathsf{PK} = \left\{\mathsf{pk}^{\mathcal{KI}}, \mathsf{pk}^{\mathcal{GU}}, \mathsf{pk}^{\mathcal{GM}}\right\}$.
5. $(\mathsf{cert}_i, (\mathsf{sk}_i, \mathsf{st}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{GM}})) \leftarrow \mathsf{JoinUser}^{\mathsf{Valid}}\,\langle U_i\ (\mathrm{ID}_i),\ \mathcal{KI}(\mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}),$ $\mathcal{GM}(\mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}})\rangle\ (\mathsf{pp}, \mathsf{PK})$. *An interactive protocol run between a user, $\mathcal{KI}$, and $\mathcal{GM}$. User participates with a public identity ID_i that can be validated according to a predicate* Valid*. $\mathcal{KI}$ and $\mathcal{GM}$ participate with their secret keys and their states. At the end of the protocol, the user gets a secret member key sk_i for a member identity ID_i, the public output is a certificate cert_i that is added to key issuer and group manager's states.*
6. $\pi \leftarrow \mathsf{MemberAuth}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \mathsf{cert}_i, \mathsf{sk}_i)$. *The user U_i executes* MemberAuth *to create an authorization for a message $\mathsf{m} \in \mathcal{M}$. On input the parameters* pp*, public keys* PK*, the message* m*, the secret key sk_i, and the associated certificate cert_i, it outputs a proof of membership π that proves his eligibility to produce* m*.*

7. $b \leftarrow \mathsf{AuthVrfy}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi)$. *On input parameters* pp*, public keys* PK*, a message* m*, a proof of membership* π *outputs a bit* b *indicating whether the message is authorized.*
8. $\mathsf{req} \leftarrow \mathsf{ReqDeanon}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{sk}^{\mathcal{GM}})$. *On input the parameters* pp*, the public keys* PK*, a message* m*, a proof of membership* π*, and the secret key of the group manager* $\mathsf{sk}^{\mathcal{GM}}$ *produces a request* req *to de-anonymize the member who posted* m.
9. $b \leftarrow \mathsf{JudgeReq}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{req})$. *On input the parameters* pp*, the public keys* PK*, a message* m*, a membership proof* π *and a request* req*, it outputs a bit* b *indicating whether* $\mathcal{GM}$ *produced the request.*
10. $\perp/\mathsf{access} \leftarrow \mathsf{GrantDeanon}\langle \mathsf{C}_\ell(\mathsf{sk}_\ell^{\mathcal{GU}})\rangle_{\mathsf{C}_l \in \mathcal{GU}}$ (pp,PK,req,m,π). *This is an interactive protocol between the guardians* $\mathcal{GU}$*. Guardian* C_l *has as secret input its secret key* $\mathsf{sk}_l^{\mathcal{GU}}$*, and all parties have common input the parameters* pp*, the public keys* PK*, a message* m*, a membership proof* π*, and a request* req*; the result of the protocol is a common output, which is either a value* access *that will allow group managerto de-anonymize the message or* $\perp$.
11. $\perp/(\mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}}) \leftarrow \mathsf{Deanon}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{sk}^{\mathcal{GM}})$. *On input the parameters* pp*, the public keys* PK*, a message* m*, a membership proof* π*, a string* access*, and the secret key of the group manager* $\mathsf{sk}^{\mathcal{GM}}$ *outputs* $\mathsf{cert}_{\mathrm{ID}}$ *and a publicly verifiable proof* $\mathsf{proof}_{\mathrm{ID}}$ *that this was the* $\mathsf{cert}_{\mathrm{ID}}$ *accountable for* m*. Otherwise, it outputs the special symbol* $\perp$.
12. $b \leftarrow \mathsf{Judge}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$ *On input the public parameters* pp*, the public keys* PK*, and information about the authenticated message (*m, π*), the string* access *from the guardians and a pair* $(\mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$ *from the group manager; it outputs a bit denoting whether the user assigned with* $\mathsf{cert}_{\mathrm{ID}}$ *is indeed accountable for the pair (*m, π*).*

4.1 Threat Model and Security Properties

For each security property of MAL, we describe a security game where the power of the adversary is modeled via oracles. The oracles and their behavior are described below.

- $\mathsf{OHonJoin}(\mathrm{ID}_i)$: When the adversary queries the oracle, an honest user is enrolled in the system. Formally, the oracle executes JoinUser for a user with identity ID_i, and outputs cert_i to the adversary; it maintains a set $\mathcal{U}_h$ which is updated with $(\mathsf{cert}_i, \mathsf{token}_i)$.
- $\mathsf{OMalJoin}(\mathrm{ID}_i, \mathsf{vk}_i)$: When queried, this oracle allows the adversary to create and control a malicious user by interacting with the adversary in protocol JoinUser, participating as key issuer. It maintains a set $\mathcal{U}_m$ of malicious users containing cert_i.
- OGuKeyGen: This is an one-time oracle that when queried the adversary cooperates in the protocol to generate the keys of the guardians. The guardians the adversary controls are denoted as $\mathcal{GU}_{\mathrm{cor}}$ and the oracle plays on behalf of the remaining, denoted $\mathcal{GU}_{\mathrm{hon}}$. Formally, this oracle consists of running $\left(\mathsf{pk}^{\mathcal{GU}}, \mathsf{st}_{\mathcal{A}}, (\mathsf{sk}_\ell^{\mathcal{GU}})_{\mathsf{C}_\ell \in \mathcal{GU}_{\mathrm{hon}}}\right) \leftarrow \mathsf{KeyGenGu}(\mathcal{A}(\cdot))\, \langle \mathsf{C}_\ell(\cdot)\rangle_{\mathsf{C}_l \in \mathcal{GU}}\, (\mathsf{pp})$.
- $\mathsf{OGetGuKeys}(i)$: When queried on input i, the adversary receives the secret key of the i-th guardian, which is added to $\mathcal{GU}_{\mathrm{cor}}$ set.
- $\mathsf{OMemberAuth}(\mathsf{m}, \mathsf{cert}_i)$: Upon query, outputs membership proof $\pi \leftarrow \mathsf{MemberAuth}\ (\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \mathsf{cert}_i, \mathsf{sk}_i)$, produced by honest user with $(\mathsf{cert}_i, \cdot) \in \mathcal{U}_h$ for message $\mathsf{m} \in \mathcal{M}$. If the user is not honest, the oracle returns $\perp$. A set $\mathcal{T}$ is updated with $(\mathsf{m}, \pi, \mathsf{cert}_i)$.
- $\mathsf{OReq}(\mathsf{m}, \pi)$: The oracle returns $\mathsf{req} \leftarrow \mathsf{ReqDeanon}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{sk}^{\mathcal{GM}})$; a set $\mathcal{Q}$ is updated with $(\mathsf{m}, \pi, \mathsf{req})$.
- $\mathsf{OGrant}(\mathsf{m}, \pi, \mathsf{req})$: When queried, a GrantDeanon protocol execution starts, i.e.,

$$\mathsf{GrantDeanon}(\mathcal{A}(\cdot)) \left\langle \mathsf{C}_\ell(\mathsf{sk}_\ell^{\mathcal{GU}}) \right\rangle_{\mathsf{C}_l \in \mathcal{GU}} (\mathsf{pp}, \mathsf{PK}, \mathsf{req}, \mathsf{m}, \pi)$$

  a set $\mathcal{G}$ is updated with $(\mathsf{m}, \pi, \mathsf{req}, \mathsf{access})$.
- $\mathsf{ODeanon}(\mathsf{m}, \pi, \mathsf{access})$: this oracle models the ability of an adversary to learn the identity of the creator of a message. When queried, the oracle runs $\mathsf{Deanon}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{sk}^{\mathcal{GM}})$; a set $\mathcal{D}$ is updated with each query with $(\mathsf{m}, \pi, \mathsf{access}, \cdot)$ where $\cdot$ denotes the value returned by the oracle.

$\underline{\mathsf{Completeness}_{\Pi}(1^{\lambda}, \mathsf{m}, \mathrm{ID})}$

- $\mathsf{pp} \leftarrow \mathsf{SetupParams}(1^{\lambda})$
- $(\mathsf{pk}^{\mathcal{KI}}, \mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}) \leftarrow \mathsf{KeyGenIssuer}(\mathsf{pp})$
- $(\mathsf{pk}^{\mathcal{GM}}, \mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}}) \leftarrow \mathsf{KeyGenManager}(\mathsf{pp})$
- $\left(\mathsf{pk}^{\mathcal{GU}}, (\mathsf{sk}_l^{\mathcal{GU}})_{\mathsf{C}_l \in \mathcal{GU}}\right) \leftarrow \mathsf{KeyGenGu}\left\langle \mathsf{C}_\ell(\cdot)\right\rangle_{\mathsf{C}_l \in \mathcal{GU}}(\mathsf{pp})$
- $\mathsf{PK} := \left\{\mathsf{pk}^{\mathcal{KI}}, \mathsf{pk}^{\mathcal{GM}}, \mathsf{pk}^{\mathcal{GU}}\right\}$
- $(\mathsf{cert}, (\mathsf{sk}, \mathsf{st}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{GM}})) \leftarrow \mathsf{JoinUser}^{\mathsf{Valid}}$
  $\left\langle U(\mathrm{ID}), \mathcal{KI}(\mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}), \mathcal{GM}(\mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}})\right\rangle(\mathsf{pp}, \mathsf{PK})$
- $\pi \leftarrow \mathsf{MemberAuth}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \mathsf{cert}, \mathsf{sk})$
- $\mathsf{req} \leftarrow \mathsf{ReqDeanon}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{sk}^{\mathcal{GM}})$
- $\mathsf{access} \leftarrow \mathsf{GrantDeanon}\left\langle \mathsf{C}_\ell(\mathsf{sk}_\ell^{\mathcal{GU}})\right\rangle_{\mathsf{C}_l \in \mathcal{GU}}(\mathsf{pp}, \mathsf{PK}, \mathsf{req}, \mathsf{m}, \pi)$
- $(\mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}}) \leftarrow \mathsf{Deanon}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{sk}^{\mathcal{GM}})$
- The output of the experiment is 1 if
  - $\mathsf{AuthVrfy}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi) = 1$
  - $\mathsf{JudgeReq}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{req}) = 1$
  - $\mathsf{Judge}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}}) = 1$
  - $\mathsf{cert} = \mathsf{cert}_{\mathrm{ID}}$

Fig. 1. $\mathsf{Completeness}_{\Pi}(1^{\lambda}, \mathsf{m}, \mathrm{ID})$

Completeness. If all parties behave honestly (i) any user with a *valid* ID can join the system and create membership authorizations that verifies; (ii) the group manager will be able to create de-anonymization requests that verify and, in collaboration with a majority of the guardians, will also manage to de-anonymize any message; and (iii) the de-anonymization process will lead to the creator of the message and the group manager will be capable to create a proof of identity that verifies. Completeness holds if the output of the experiment of Fig. 1 is 1:

Definition 2 (Completeness). *A* MAL *scheme Π satisfies* completeness *if, for any message $\mathsf{m} \in \mathcal{M}$ and identity* ID*,* $\Pr\left[\mathsf{Completeness}_{\Pi}(1^{\lambda}, \mathsf{m}, \mathrm{ID}) = 1\right] = 1$.

Unforgeability captures the property that anyone (even group manager and guardians) who is *not enrolled* in the system (i.e., has not executed protocol JoinUser) cannot produce valid membership authorizations. We capture this in a game where an adversary $\mathcal{A}$ controls group manager and guardians, and has access to oracles OHonJoin(ID) and OMemberAuth, meaning it can create new users (but not to *control* them) and to see honestly generated membership authorizations of her choice. The adversary wins the game if she can produce a pair (m, π) that verifies, without controlling any user and without querying m in the OMemberAuth oracle. The game is presented in Fig. 2.

Definition 3 (Unforgeability). *A* MAL *scheme Π satisfies* unforgeability *if for any PPT adversary $\mathcal{A}$, there is a negligible function μ such that*

$$\Pr\left[\mathsf{Exp\text{-}Unforge}_{\Pi,\mathcal{A}}(1^{\lambda}) = 1\right] \leq \mu(\lambda).$$

User non-frameability demands that no one can falsely accuse a user U_i of being the creator of a certain value. Even if all guardians, the manager and some other users are malicious and colluding, user U_i cannot be framed (if $\mathcal{KI}$ is honest). Formally, the adversary controls the group manager, the guardians, and malicious users; she wins if she can provide a valid pair (m, π) that opens to the identity of an honest user. This property is shown in Fig. 3.

Definition 4. *(*User Non-frameability*) A* MAL *scheme Π satisfies user non-frameability if for all PPT adversaries $\mathcal{A}$ there exists a negligible function μ such that* $\Pr\left[\mathsf{Exp\text{-}UserNonFrame}_{\Pi,\mathcal{A}}(1^{\lambda}) = 1\right] \leq \mu(\lambda)$.

$\underline{\mathsf{Exp\text{-}Unforge}_{\Pi,\mathcal{A}}(1^\lambda)}$
$\mathcal{A}$ has access to $\mathcal{O} = \{\mathsf{OHonJoin}, \mathsf{OMemberAuth}\}$. $\mathcal{A}$ fully controls the group manager $\mathcal{GM}$, and the guardians $\mathcal{GU}$.

- Initialization: (1) Parameters $\mathsf{pp} \leftarrow \mathsf{SetupParams}(1^\lambda)$. (2) Sets for the oracles: $\mathcal{U}_h, \mathcal{T} := \emptyset$. (3) key issuer's keys: $(\mathsf{pk}^{\mathcal{KI}}, \mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}) \leftarrow \mathsf{KeyGenIssuer}(\mathsf{pp})$.
- $(\mathsf{pk}^{\mathcal{GM}}, \mathsf{pk}^{\mathcal{GU}}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}^{\mathcal{KI}})$; $\mathsf{PK} := \{\mathsf{pk}^{\mathcal{GM}}, \mathsf{pk}^{\mathcal{GU}}, \mathsf{pk}^{\mathcal{KI}}\}$.
- $(\mathsf{m}, \pi) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{st}^{\mathcal{A}})$.
- The output of the experiment is 1 iff $\mathsf{AuthVrfy}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi) = 1$, and $(\mathsf{m}, \pi, \cdot) \notin \mathcal{T}$ (the proof π was not computed by the challenger).

Fig. 2. $\mathsf{Exp\text{-}Unforge}_{\Pi,\mathcal{A}}$

$\underline{\mathsf{Exp\text{-}UserNonFrame}_{\Pi,\mathcal{A}}\ (1^\lambda)}$
$\mathcal{A}$ has access to $\mathcal{O} = \{\mathsf{OHonJoin}, \mathsf{OMemberAuth}, \mathsf{OMalJoin}\}$. $\mathcal{A}$ fully controls the group manager $\mathcal{GM}$, and the guardians $\mathcal{GU}$.

- Initialization: (1) Parameters $\mathsf{pp} \leftarrow \mathsf{SetupParams}(1^\lambda)$. (2) Sets for the oracles: $\mathcal{U}_h, \mathcal{T}, \mathcal{U}_m := \emptyset$. (3) key issuer's keys: $(\mathsf{pk}^{\mathcal{KI}}, \mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}) \leftarrow \mathsf{KeyGenIssuer}(\mathsf{pp})$.
- $(\mathsf{pk}^{\mathcal{GM}}, \mathsf{pk}^{\mathcal{GU}}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}^{\mathcal{KI}})$; $\mathsf{PK} := \{\mathsf{pk}^{\mathcal{KI}}, \mathsf{pk}^{\mathcal{GU}}, \mathsf{pk}^{\mathcal{GM}}\}$.
- $(\mathsf{m}, \pi, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{st}^{\mathcal{A}})$.
- The output of the experiment is 1 iff
  - $\mathsf{AuthVrfy}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi) = 1$, and
  - $(\mathsf{m}, \pi, \cdot) \notin \mathcal{T}$ (the proof π was not computed by the challenger), and
  - $\mathsf{Judge}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}}) = 1$ and,
  - $(\mathsf{cert}_{\mathrm{ID}}, \cdot) \in \mathcal{U}_h$.

Fig. 3. $\mathsf{Exp\text{-}UserNonFrame}_{\Pi,\mathcal{A}}$

User accountability captures the fact that, when de-anonymized, a pair (m, π) must trace back to *some* user. $\mathcal{A}$'s goal is to create a pair (m, π) that is not correctly de-anonymized, while having control over malicious users and the guardians. It can also ask for honest users' membership proofs, requests and de-anonymizations. The formal game for user accountability is in Fig. 4.

Definition 5 (User Accountability). *A* MAL *scheme Π has* User Accountability *if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function μ, such that* $\Pr[\mathsf{t\text{-}Exp\text{-}UserAccountability}_{\Pi,\mathcal{A}}(1^\lambda) = 1] \leq \mu(\lambda)$.

Manager Non-Frameability guarantees that no one can accuse $\mathcal{GM}$ of creating a de-anonymization request, even if all guardians and users are malicious and collude. The adversary has full control of users and guardians and can ask for requests/deanonymizations; her goal is to craft a de-anonymization request that verifies. The formal game is in Fig. 5.

Definition 6. *(*Manager Non-Frameability*) A* MAL *scheme Π has* Manager Non-Frameability *if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function μ, such that* $\Pr[\mathsf{Exp\text{-}ManagerNonFrame}_{\Pi,\mathcal{A}}(1^\lambda) = 1] \leq \mu(\lambda)$.

Anonymity guarantees that no one can extract information about the creator of a pair (m, π), unless $\mathcal{GM}$ and the guardians collaborate. To capture this, we consider two cases: (1) $\mathcal{GM}$ and a minority of the guardians are corrupted, (2) only the guardians are corrupted. In both, the anonymity property is captured by an indistinguishability definition. The formal games are presented in Fig. 6,7

$\underline{\mathsf{t\text{-}Exp\text{-}UserAccountability}_{\Pi,\mathcal{A}}(1^\lambda)}$
$\mathcal{A}$ has access to $\mathcal{O} = \{\mathsf{OHonJoin}, \mathsf{OMemberAuth}, \mathsf{OMalJoin}, \mathsf{OReq}, \mathsf{OGuKeyGen}, \mathsf{OGetGuKeys}, \mathsf{ODeanon}\}$. Namely, $\mathcal{A}$ can create malicious users, control t malicious guardians, and learn the keys of honest guardians.

- Initialization: (1) Parameters $\mathsf{pp} \leftarrow \mathsf{SetupParams}(1^\lambda)$; (2) Public Keys $(\mathsf{pk}^{\mathcal{KI}}, \mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}) \leftarrow \mathsf{KeyGenIssuer}(\mathsf{pp})$; $(\mathsf{pk}^{\mathcal{GM}}, \mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}}) \leftarrow \mathsf{KeyGenManager}(\mathsf{pp})$; (3) Oracle Sets: $\mathcal{U}_h, \mathcal{U}_m, \mathcal{Q}, \mathcal{G}, \mathcal{D} := \emptyset$.
- $(\mathsf{m}, \pi) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{pk}^{\mathcal{GM}}, \mathsf{pk}^{\mathcal{GU}}, \mathsf{pk}^{\mathcal{KI}})$.
- The output of the experiment is 1 iff
  - $\mathsf{AuthVrfy}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi) = 1$
  - $\mathsf{req} \leftarrow \mathsf{ReqDeanon}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{sk}^{\mathcal{GM}})$
  - $\mathsf{access} \leftarrow \mathsf{GrantDeanon}(\mathcal{A}(\mathsf{st}_{\mathcal{A}}))\ \langle \mathsf{C}_\ell(\mathsf{sk}_\ell^{\mathcal{GU}}) \rangle_{\mathsf{C}_l \in \mathcal{GU}_{\mathrm{hon}}}\ (\mathsf{pp}, \mathsf{PK}, \mathsf{req}, \mathsf{m}, \pi)$
  - $z \leftarrow \mathsf{Deanon}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{sk}^{\mathcal{GM}})$
  - $\forall(\mathsf{cert}_{\mathrm{ID}}, \cdot) \in \mathcal{U}_h \cup \mathcal{U}_m,\ z \neq (\mathsf{cert}_{\mathrm{ID}}, \cdot)$

Fig. 4. $\mathsf{t\text{-}Exp\text{-}UserAccountability}_{\Pi,\mathcal{A}}$

$\underline{\mathsf{Exp\text{-}ManagerNonFrame}_{\Pi,\mathcal{A}}(1^\lambda)}$
$\mathcal{A}$ has access to the oracle $\mathcal{O} = \{\mathsf{OHonJoin}, \mathsf{OMalJoin}, \mathsf{OMemberAuth}, \mathsf{OReq}, \mathsf{ODeanon}\}$
The experiment works as follows:

- Initialization: (1) Parameters $\mathsf{pp} \leftarrow \mathsf{SetupParams}(1^\lambda)$; (2) Public Keys $(\mathsf{pk}^{\mathcal{KI}}, \mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}) \leftarrow \mathsf{KeyGenIssuer}(\mathsf{pp})$; $(\mathsf{pk}^{\mathcal{GM}}, \mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}}) \leftarrow \mathsf{KeyGenManager}(\mathsf{pp})$; (3) Oracle Sets: $\mathcal{U}_h, \mathcal{U}_m, \mathcal{Q}, \mathcal{G}, \mathcal{D} := \emptyset$
- $(\mathsf{pk}^{\mathcal{GU}}, \mathsf{m}, \pi, \mathsf{req}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{pk}^{\mathcal{KI}}, \mathsf{pk}^{\mathcal{GM}})$
- The output of the experiment is 1 iff
  - $\mathsf{JudgeReq}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{req}) = 1$
  - $(\mathsf{m}, \pi, \mathsf{req}) \notin \mathcal{Q}$

Fig. 5. $\mathsf{Exp\text{-}ManagerNonFrame}_{\Pi,\mathcal{A}}(1^\lambda)$

Remark 1. Anonymity also implicitly covers a property of *manager accountability*: manager cannot open messages on its own, it must ask for permission and if it wants to blame a user for a message, needs to present a proof.

Definition 7 (Anonymity). *A* MAL *scheme has* t-Anonymity *if, for all PPT adversaries* $\mathcal{A}_1, \mathcal{A}_2$*, there exists a negligible function* μ *such that both the following hold:*

$$\Pr\left[\mathsf{Exp\text{-}UAnon\&MalGM}_{\Pi,\mathcal{A}_1}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mu(\lambda),$$

$$\Pr\left[\mathsf{t\text{-}Exp\text{-}UAnon\&MalGuard}_{\Pi,\mathcal{A}_2}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mu(\lambda).$$

5 Instantiation of MAL

We describe an instantiation Π-MAL of MAL in three phases: generation of keys, creating and verifying membership proofs, and de-anonymization, presented in Figures 8, 9 and 10 respectively.

The security primitives underlying our protocols are a one-way function f, a signature scheme S, a public key encryption scheme E, a threshold public key encryption scheme TE with *decryption simulatability*, a succinct simulation-extractable non-interactive zero-knowledge argument (nizk), and a (simulation-extractable) succinct nizk [28] (snark). At a high-level, the scheme works as described in Sec. 1.

$\mathsf{Exp\text{-}UAnon\&MalGM}_{\Pi,\mathcal{A}}(1^\lambda)$

$\mathcal{A}$ has to the oracles $\mathcal{O} = \{\mathsf{OHonJoin}, \mathsf{OMalJoin}, \mathsf{OMemberAuth}, \mathsf{OGuKeyGen}, \mathsf{OGetGuKeys}, \mathsf{OGrant}\}$.

- Initialization: (1) Parameters $\mathsf{pp} \leftarrow \mathsf{SetupParams}(1^\lambda)$; (2) Public Keys $(\mathsf{pk}^{\mathcal{KI}}, \mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}) \leftarrow \mathsf{KeyGenIssuer}(\mathsf{pp})$; (3) Oracle Sets: $\mathcal{U}_h, \mathcal{U}_m, \mathcal{T}, \mathcal{G} := \emptyset$
- $(\mathsf{pk}^{\mathcal{GM}}, \mathsf{m}, \mathsf{cert}_0, \mathsf{cert}_1) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{pk}^{\mathcal{KI}})$
- The challenger checks:
  - If $(\mathsf{cert}_0, \cdot)$ or $(\mathsf{cert}_1, \cdot) \notin \mathcal{U}_h$ output $\perp$.
  - Else, let $\mathsf{token}_0, \mathsf{token}_1$ be the corresponding secret states. Compute $b \leftarrow \{0,1\}$; $\pi \leftarrow \mathsf{MemberAuth}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \mathsf{cert}_b, \mathsf{sk}_b)$.
- $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{st}, \pi)$
- The output of the experiment is 1 iff $(b = b') \wedge (\mathsf{m}, \pi, \cdot, \cdot) \notin \mathcal{G} \wedge |\mathcal{GU}_{\text{cor}}| \leq t$

Fig. 6. $\mathsf{Exp\text{-}UAnon\&MalGM}_{\Pi,\mathcal{A}}(1^\lambda)$

$\mathsf{t\text{-}Exp\text{-}UAnon\&MalGuard}_{\Pi,\mathcal{A}}(1^\lambda)$

$\mathcal{A}$ has access to the oracles $\mathcal{O} = \{\mathsf{OHonJoin}, \mathsf{OMemberAuth}, \mathsf{OMalJoin}, \mathsf{OReq}, \mathsf{ODeanon}\}$

- Initialization: (1) Parameters $\mathsf{pp} \leftarrow \mathsf{SetupParams}(1^\lambda)$; (2) Public Keys $(\mathsf{pk}^{\mathcal{KI}}, \mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}) \leftarrow \mathsf{KeyGenIssuer}(\mathsf{pp})$; $(\mathsf{pk}^{\mathcal{GM}}, \mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}}) \leftarrow \mathsf{KeyGenManager}(\mathsf{pp})$. (4) Oracle Sets: $\mathcal{U}_h, \mathcal{U}_m, \mathcal{T}, \mathcal{Q}, \mathcal{D} := \emptyset$
- $(\mathsf{pk}^{\mathcal{GU}}, \mathsf{m}, \mathsf{cert}_0, \mathsf{cert}_1) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{pk}^{\mathcal{KI}}, \mathsf{pk}^{\mathcal{GM}})$
- The challenger checks:
  - If $(\mathsf{cert}_0, \cdot)$ or $(\mathsf{cert}_1, \cdot) \notin \mathcal{U}_h$ output $\perp$.
  - Else, let $\mathsf{token}_0, \mathsf{token}_1$ be the corresponding secret keys. Compute $b \leftarrow \{0,1\}$; $\pi \leftarrow \mathsf{MemberAuth}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \mathsf{cert}_b, \mathsf{sk}_b)$.
- $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{st}, \pi)$
- The output of the experiment is 1 iff: $(b = b') \wedge (\mathsf{m}, \pi, \cdot, \cdot) \notin \mathcal{D}$

Fig. 7. $\mathsf{t\text{-}Exp\text{-}UAnon\&MalGuard}_{\Pi,\mathcal{A}}(1^\lambda)$

First, the setup parameters are generated. Using these public parameters, the key issuer, the group manager, and the guardians create their keys. $\mathcal{KI}$ creates a signing key pair, $\mathcal{GM}$ creates a public key encryption key pair and signing key pair, and the guardians execute the threshold encryption scheme to create their secret keys and a common public key.

To join the platform, a user needs to receive a certificate from $\mathcal{KI}$. To do so, a user samples a random value sk from the domain of a one-way function f and sends $\mathsf{vk} \leftarrow f(\mathsf{sk})$ and her "real" identity ID to $\mathcal{KI}$. Upon verification of ID, $\mathcal{KI}$ sends a signature σ on the value $(\text{ID}\|\mathsf{vk})$ to the user. The user's certificate is the tuple $\mathsf{cert} := (\text{ID}, \mathsf{vk}, \sigma)$. $\mathcal{KI}$ keeps a list of the users as $\mathsf{Members}$. Multiple lists can be initialized if multiple applications use the same key issuer.

To anonymously post a message m, the user proves she is authorized by creating a membership proof (c_2, proof). The first item, c_2, is a valid encryption under the public key of the guardians of a ciphertext c_1, which itself is an encryption under the public key of $\mathcal{GM}$ of a certificate $\mathsf{cert} := (\text{ID}, \mathsf{vk}, \sigma)$. The second item, proof, must be a proof of well formation of the ciphertext. To prove c_2 is well formed, the user produces proof, which is a succinct non-interactive zero-knowledge proof of knowledge of the secret key associated to the public key vk contained in cert.

To de-anonymize a tuple $(\mathsf{m}, \mathsf{proof})$, $\mathcal{GM}$ signs $(\mathsf{m}, \mathsf{proof})$ and broadcasts the signature. The guardians first assert that the request was made by the group manager, then publish the request[9] and proceed to the

[9] Application dependent conditions can be additionally included, e.g., rules on which messages should be de-anonymized.

threshold decryption protocol. The output of such protocol is the decryption of c_2 (i.e., the value c_1) along with a proof that the decryption is valid. $\mathcal{GM}$ can then use his secret key to decrypt c_1 and retrieve the certificate cert associated to the pair $(\mathsf{m}, \mathsf{proof})$.

To achieve public verifiability of an opening, $\mathcal{GM}$ provides a nizk proof of knowledge that a specific user's certificate cert was the output of decrypting the ciphertext c_1.

Zero-Knowledge Relation. The relation $\mathcal{R}$ for a statement $(\mathsf{m}, c_2, \mathsf{PK})$ is described below:

$$\mathcal{R} = \left\{ \begin{array}{c} ((\mathsf{m}, c_2, \mathsf{PK}), (r_2, c_1, r_1, \mathsf{cert}_i, \mathsf{sk}_i)) \text{ s.t. } \mathsf{PK} = ((\mathsf{vk}_\mathsf{S}^{\mathcal{KI}}), (\mathsf{pk}_\mathsf{E}^{\mathcal{GM}}, \mathsf{vk}_\mathsf{S}^{\mathcal{GM}}), (\mathsf{pk}_\mathsf{TE}^{\mathcal{GU}})) \\ \wedge \mathsf{cert}_i = (\mathsf{vk}_i, \mathrm{ID}_i, \sigma_i) \wedge \mathsf{S.Vrfy}_{\mathsf{vk}_\mathsf{S}^{\mathcal{KI}}}(\mathrm{ID}_i \| \mathsf{vk}_i, \sigma_i) = 1 \\ \wedge c_1 = \mathsf{E.Enc}_{\mathsf{pk}_\mathsf{E}^{\mathcal{GM}}}(\mathsf{cert}_i; r_1) \wedge \mathsf{vk}_i = f(\mathsf{sk}_i) \wedge c_2 = \mathsf{TE.Enc}_{\mathsf{pk}_\mathsf{TE}^{\mathcal{GU}}}(\mathsf{m} \| c_1; r_2) \end{array} \right\}$$

We also use a nizk proof for the relation $\mathcal{VD} = \{(\mathsf{pk}, \mathsf{m}, c), (\mathsf{sk}) \text{ s.t. } \mathsf{m} = \mathsf{E.Dec}_{\mathsf{sk}}(c)\}$, for correct decryption.

$$\begin{aligned} \mathcal{R} = \Big\{ (\mathsf{m}, c_2, \mathsf{vk}_{\mathcal{GM}}, \mathsf{pk}^{\mathcal{GU}}, \mathsf{pk}^{\mathcal{GM}}, \mathsf{vk}^{\mathcal{KI}}), (r_2, c_1, r_1, \mathsf{cert}_i, \mathsf{sk}_i, r_s) \text{ s.t.} \\ \mathsf{pk}^{\mathcal{GU}} = \mathsf{pk}_{\mathsf{TE}} \wedge \mathsf{cert}_i = (\mathsf{vk}_i, \mathrm{ID}_i, \sigma_i) \wedge \\ \mathsf{S.Vrfy}_{\mathsf{vk}^{\mathcal{KI}}}(\mathrm{ID}_i \| \mathsf{vk}_i) = 1 \wedge c_2 = \mathsf{TE.Enc}_{\mathsf{pk}_{\mathsf{TE}}}(\mathsf{m} \| c_1; r_2) \wedge \\ c_1 = \mathsf{E.Enc}_{\mathsf{pk}_\mathsf{E}^{\mathcal{GM}}}(\mathsf{cert}_i; r_1) \wedge (\mathsf{vk}_i, \mathsf{sk}_i) := \mathsf{S.KeyGen}(r_s) \Big\} \end{aligned}$$

Efficiency. We discuss the costs of Π-MAL when instantiated with concrete primitives. We use ElGamal for the IND-CPA encryption scheme, Threshold ElGamal [32] with a verifiable secret sharing scheme for key generation (e.g., [33]), the Chaum-Pedersen [34] protocol for verifiable decryption, succinct non-interactive argument of Groth-Maller [28] for relation $\mathcal{R}$, ECDSA digital signature [35] for the signature scheme, and SHA-256 [36] for the one-way function on a domain defined by the security parameter. We also use SHA-256 to instantiate the random oracle for the non-interactive version of the Chaum-Pedersen protocol.

The cost of user authentication is dominated by the cost of computing a SNARK proof. Generating the SNARK proof depends on the complexity of the verification circuit for the underlying relation $\mathcal{R}$. .

The work of the guardians is dominated by setting up and executing threshold ElGamal. Threshold ElGamal uses a verifiable secret sharing scheme, so each party must non-interactively send a constant number of messages to every other party (thus in total $\mathcal{O}(n^2)$ messages are sent). Each message is $\mathcal{O}(\lambda)$ – constant in the size of the share. The cost of producing/processing these messages is dominated by $\mathcal{O}(n)$ group operations per party. To decrypt a message, each guardian broadcasts their decryption share along with a nizk proof that their decryption is consistent with their secret share (a Chaum-Pedersen proof). The guardians must verify the zk proofs from the others and get at least $t + 1$ valid proofs. Each verification requires a SHA-256 computation and a constant number of group operations. Now, given at least $t + 1$ decryption shares, the guardians can decrypt the message by doing $\mathcal{O}(t)$ group operations. Thus, verifying the output of the guardians requires $\mathcal{O}(t)$ SHA-256 computations and $\mathcal{O}(t)$ group operations overall. Since the de-anonymization activities are supposed to be sporadic and are done off-line, the communication and computational cost incurred in this operation is not a bottleneck for deployment.

Finally, the complexity of the judge procedures only requires verification of the correctness of the two layers of encryption. This boils down to the verification of $t + 2$ Chaum-Pedersen proofs and $\mathcal{O}(t)$ group operations. for the inner, it just needs to verify one Chaum-Pedersen proof. In summary, the only on-line operation done by the user when computing a membership proof is a snark computation. All guardians' operations can be performed off-line, and hence are not a bottleneck for the system.

5.1 Security

In this sections we present the theorems that capture the security property of the construction and give some intuition for the proofs. The full proofs can be found in App. B.

$\underline{\mathsf{SetupParams}}(1^\lambda)$
- $\mathsf{pp}_\mathsf{E} \leftarrow \mathsf{E.Setup}(1^\lambda)$, $\mathsf{pp}_\mathsf{S} \leftarrow \mathsf{S.Setup}(1^\lambda)$; $\mathsf{pp}_\mathsf{TE} \leftarrow \mathsf{TE.Setup}(1^\lambda)$
- $(\mathsf{crs}_\mathcal{R}, \tau_\mathcal{R}) \leftarrow \mathsf{snark}_\mathcal{R}.\mathsf{Setup}(1^\lambda)$
- $(\mathsf{crs}_{\mathcal{VD}}, \tau_{\mathcal{VD}}) \leftarrow \mathsf{NIZK}_{\mathcal{VD}}.\mathsf{Setup}(1^\lambda)$
- Output $\mathsf{pp} := (\mathsf{pp}_\mathsf{E}, \mathsf{pp}_\mathsf{S}, \mathsf{pp}_\mathsf{TE}, \mathsf{crs}_\mathcal{R}, \mathsf{crs}_{\mathcal{VD}})$

$\underline{\mathsf{KeyGenIssuer}}(\mathsf{pp})$
- Parse pp as $(\mathsf{pp}_\mathsf{E}, \mathsf{pp}_\mathsf{S}, \mathsf{pp}_\mathsf{TE}, \mathsf{crs}_\mathcal{R}, \mathsf{crs}_{\mathcal{VD}})$
- $(\mathsf{vk}_\mathsf{S}^{\mathcal{KI}}, \mathsf{sk}_\mathsf{S}^{\mathcal{KI}}) \leftarrow \mathsf{S.KeyGen}(\mathsf{pp}_\mathsf{S})$.
- Initialize an empty list $\mathsf{Members}$
- Output $(\mathsf{vk}_\mathsf{S}^{\mathcal{KI}}, \mathsf{sk}_\mathsf{S}^{\mathcal{KI}}, \mathsf{Members})$

$\underline{\mathsf{KeyGenManager}}(\mathsf{pp})$
- Parse pp as $(\mathsf{pp}_\mathsf{E}, \mathsf{pp}_\mathsf{S}, \mathsf{pp}_\mathsf{TE}, \mathsf{crs}_\mathcal{R}, \mathsf{crs}_{\mathcal{VD}})$
- $(\mathsf{pk}_\mathsf{E}^{\mathcal{GM}}, \mathsf{sk}_\mathsf{E}^{\mathcal{GM}}) \leftarrow \mathsf{E.KeyGen}(\mathsf{pp}_\mathsf{E})$;
  $(\mathsf{vk}_\mathsf{S}^{\mathcal{GM}}, \mathsf{sk}_\mathsf{S}^{\mathcal{GM}}) \leftarrow \mathsf{S.KeyGen}(\mathsf{pp}_\mathsf{S})$
- Initialize an empty list $\mathsf{Members}$
- Set $\mathsf{pk}^{\mathcal{GM}} := (\mathsf{pk}_\mathsf{E}^{\mathcal{GM}}, \mathsf{vk}_\mathsf{S}^{\mathcal{GM}})$; $\mathsf{sk}^{\mathcal{GM}} := (\mathsf{sk}_\mathsf{E}^{\mathcal{GM}}, \mathsf{sk}_\mathsf{S}^{\mathcal{GM}})$; $\mathsf{st}^{\mathcal{GM}} = \mathsf{Members}$
- Output $(\mathsf{sk}^{\mathcal{GM}}, \mathsf{pk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}})$

$\underline{\mathsf{KeyGenGu}}\ \langle \mathsf{C}_\ell(\cdot)\rangle_{\mathsf{C}_l \in \mathcal{GU}}\ (\mathsf{pp})$
- Parse pp as $(\mathsf{pp}_\mathsf{E}, \mathsf{pp}_\mathsf{S}, \mathsf{pp}_\mathsf{TE}, \mathsf{crs}_\mathcal{R}, \mathsf{crs}_{\mathcal{VD}})$
- Parties execute
  $\left(\mathsf{pk}_\mathsf{TE}, \left(\mathsf{sk}_\mathsf{TE}^l\right)_{\mathsf{C}_\ell \in \mathcal{GU}}\right) \leftarrow \mathsf{TE.KeyGen}\langle \mathsf{C}_\ell(\cdot)\rangle_{\mathsf{C}_\ell \in \mathcal{GU}}\ (\mathsf{pp}_\mathsf{TE})$
- $\mathsf{pk}^{\mathcal{GU}} \leftarrow \mathsf{pk}_\mathsf{TE}$ and $\mathsf{sk}_l^{\mathcal{GU}} \leftarrow \mathsf{sk}_\mathsf{TE}^l$

$\underline{\mathsf{JoinUser}}(\langle U_i(\mathrm{ID}_i), \mathcal{KI}(\mathsf{sk}^{\mathcal{KI}}, \mathsf{st}^{\mathcal{KI}}), \mathcal{GM}(\mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}})\rangle(\mathsf{pp}, \mathsf{PK})$
- Parse pp as $(\mathsf{pp}_\mathsf{E}, \mathsf{pp}_\mathsf{S}, \mathsf{pp}_\mathsf{TE}, \mathsf{crs}_\mathcal{R}, \mathsf{crs}_{\mathcal{VD}})$ and PK as $((\mathsf{vk}_\mathsf{S}^{\mathcal{KI}}), (\mathsf{pk}_\mathsf{E}^{\mathcal{GM}}, \mathsf{vk}_\mathsf{S}^{\mathcal{GM}}), (\mathsf{pk}_\mathsf{TE}^{\mathcal{GU}}))$
- U_i: Samples $\mathsf{sk}_i \leftarrow \mathcal{SK}$ and sets $\mathsf{vk}_i := f(\mathsf{sk}_i)$
- $U_i \rightarrow \mathcal{KI}$: $\mathrm{ID}_i, \mathsf{vk}_i$
- $\mathcal{KI}$: if $\mathsf{Valid}(\mathrm{ID}_i) = 1 \wedge (\mathrm{ID}_i, \cdot, \cdot) \notin \mathsf{Members} : \sigma_i \leftarrow \mathsf{S.Sign}_{\mathsf{sk}_\mathsf{S}^{\mathcal{KI}}}(\mathrm{ID}_i \| \mathsf{vk}_i)$. Else: Output $\perp$ and halt.
- $\mathcal{KI} \rightarrow U_i$: σ_i
- U_i: If $\mathsf{S.Vrfy}_{\mathsf{vk}_\mathsf{S}^{\mathcal{KI}}}(\mathrm{ID}_i \| \mathsf{vk}_i, \sigma_i) = 1$: $\mathsf{cert}_i := (\mathrm{ID}_i, \mathsf{vk}_i, \sigma_i)$. Else: Output $\perp$ and halt.
- $\mathcal{KI}$: $\mathsf{Members} := \mathsf{Members} \cup \mathsf{cert}_i$ (Send to $\mathcal{GM}$)
- U_i: Output $(\mathsf{cert}_i, \mathsf{sk}_i)$
- $\mathcal{KI}$: Output cert_i

Fig. 8. Protocol Π-MAL: Parameter setup and Key Generation Protocols and Algorithms.

Theorem 1. *Given that the signature scheme* S *satisfies unforgeability,* $\mathsf{snark}_\mathcal{R}$ *has zero-knowledge and simulation-extractability and* f *is a one-way function, the Mutual Accountability Protocol* Π-MAL *satisfies Unforgeability (Def. 3).*

Intuition. $\mathcal{A}$ wins Exp-Unforge if she produces (m, π) such that $\mathsf{AuthVrfy}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi) = 1$ and π is not the output of a query on OMemberAuth, without knowing any user's secret key. Because $\pi = (c_2, \mathsf{proof})$, $\mathcal{A}$ can provide a valid proof of knowledge proof of the secret sk associated to a $\mathsf{cert} = (\mathrm{ID}, \mathsf{vk}, \sigma)$ without executing any JoinUser procedure. Hence, $\mathcal{A}$ can (i) provide a proof that verifies but was not constructed with a valid witness (thus violating Simulation Extractability), or (ii) invert one of the vk in the pool of all honest certificates (thus violating one-wayness of f), or (iii) forge a signature σ on a public key vk that she picked (violating unforgeability of the Signature Scheme).

Theorem 2. *Given that the signature scheme* S *satisfies unforgeability,* $\mathsf{snark}_\mathcal{R}$ *has zero-knowledge,* $\mathsf{NIZK}_{\mathcal{VD}}$ *and* $\mathsf{snark}_\mathcal{R}$ *both have simulation-extractability,* TE *is a sound Threshold Signature Scheme and* f *is a one-way function, the Mutual Accountability Protocol* Π-MAL *satisfies User Non-Frameability (Def. 4).*

$\underline{\mathsf{MemberAuth}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \mathsf{cert}_i, \mathsf{sk}_i)$

- Parse pp as $(\mathsf{pp}_{\mathsf{E}}, \mathsf{pp}_{\mathsf{S}}, \mathsf{pp}_{\mathsf{TE}}, \mathsf{crs}_{\mathcal{R}}, \mathsf{crs}_{\mathcal{VD}})$
- Parse PK as $((\mathsf{vk}_{\mathsf{S}}^{\mathcal{KI}}), (\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, \mathsf{vk}_{\mathsf{S}}^{\mathcal{GM}}), (\mathsf{pk}_{\mathsf{TE}}^{\mathcal{GU}}))$
- $c_1 \leftarrow \mathsf{E.Enc}_{\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}}(\mathsf{cert}_i; r_1)$; $c_2 \leftarrow \mathsf{TE.Enc}_{\mathsf{pk}_{\mathsf{TE}}^{\mathcal{GU}}}(\mathsf{m}||c_1; r_2)$
- $\mathsf{proof} \leftarrow \mathsf{snark}_{\mathcal{R}}.\mathsf{Prove}(\mathsf{crs}, (\mathsf{m}, c_2, \mathsf{PK}), (r_2, c_1, r_1, \mathsf{cert}_i, \mathsf{sk}_i))$
- Output $\pi := (c_2, \mathsf{proof})$

$\underline{\mathsf{AuthVrfy}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi)$

- Parse pp as $(\mathsf{pp}_{\mathsf{E}}, \mathsf{pp}_{\mathsf{S}}, \mathsf{pp}_{\mathsf{TE}}, \mathsf{crs}_{\mathcal{R}}, \mathsf{crs}_{\mathcal{VD}})$
- Parse PK as $((\mathsf{vk}_{\mathsf{S}}^{\mathcal{KI}}), (\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, \mathsf{vk}_{\mathsf{S}}^{\mathcal{GM}}), (\mathsf{pk}_{\mathsf{TE}}^{\mathcal{GU}}))$
- Parse $\pi = (c_2, \mathsf{proof})$
- Output $b \leftarrow \mathsf{snark}_{\mathcal{R}}.\mathsf{Vrfy}(\mathsf{crs}, (\mathsf{m}, c_2, \mathsf{PK}), \mathsf{proof})$

Fig. 9. Protocol Π-MAL: Anonymous Membership Proof Generation and Verification Algorithms

Intuition. The only difference from Th. 1 is that $\mathcal{A}$ can create malicious users and thus the winning condition changes from being able to compute a pair (m, π) that passes verification, to providing a tuple $(\mathsf{m}, \pi, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$ that *opens* to the identity ID of an honest user. Thus, if $\mathcal{A}$ wins User Non-Frameability, it either wins Unforgeability or is able to (i) produce a valid proof $\mathsf{proof}_{\mathrm{ID}}$ for the relation $\mathcal{VD}$ without a witness (thus, breaking simulation extractability of $\mathsf{NIZK}_{\mathcal{VD}}$), or (ii) provide two different $\mathsf{cert}, \mathsf{cert}_{\mathrm{ID}}$ that are part of a valid witness for $\mathcal{R}$ and $\mathcal{VD}$, respectively (breaking soundness of the Threshold Scheme).

Theorem 3. *Construction Π-MAL satisfies User Accountability (Def. 5).*

Intuition. $\mathcal{A}$ wins if she provides a pair (m, π) that, upon going through the de-anonymization process, does not yield to a cert belonging to any user. Intuitively $\mathcal{A}$ could (i) forge a signature on the key of $\mathcal{KI}$, or (ii) create a proof of Membership without holding any valid certificate violating simulation extractability of $\mathsf{snark}_{\mathcal{R}}$.

Theorem 4. *Assuming a secure Signature Scheme S, the Protocol Π-MAL satisfies Manager Non-Frameability (Def. 6).*

Intuition. $\mathcal{A}$ wins if she can produce a valid $\mathsf{req} = \mathsf{Sign}_{\mathsf{sk}_{\mathsf{S}}^{\mathcal{GM}}}(\mathsf{m}, \pi)$ that was not outputted by OReq. Since req is a signature, $\mathcal{A}$ can be reduced to an adversary breaking the unforgeability of the Signature Scheme S.

Theorem 5. *Assuming a secure Encryption scheme E, a secure Threshold Encryption Scheme TE and secure $\mathsf{NIZK}_{\mathcal{R}}$, $\mathsf{NIZK}_{\mathcal{VD}}$, Protocol Π-MAL satisfies Anonymity (Def. 7).*

Intuition. In both cases of anonymity $\mathcal{A}$ must distinguish between membership permissions generated by different users. For both proofs, we replace real memberships by simulated ones. Then, we can extract a witness and do a reduction to Indistinguishability of Encryption either in the plain, or the threshold setting.

6 Instantiation based on t-out-of-n Group Signatures (Camenisch et al. [8])

Camenisch et al. [8] build a dynamic group signature where committees control both enrollments in the group and de-anonymization, thus distributing trust. Their scheme is based on Pointcheval-Sanders signature scheme, which features efficient zero knowledge proofs of knowledge of a verifying message-signature pair, and some sigma protocols to ensure honest behavior of the parties involved. Importantly, no generic nizk is used which makes the construction very efficient.

In this section, we introduce an adapted version of the scheme in [8] that fits our framework. Below, we first present the tools for the Camenisch et al. construction, then we give a high-level overview of the construction and the modifications that are needed to adapt their scheme to the MAL framework, and finally we describe the modified protocol.

$\underline{\mathsf{ReqDeanon}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{sk}^{\mathcal{GM}})$
- Parse pp as $(\mathsf{pp}_{\mathsf{E}}, \mathsf{pp}_{\mathsf{S}}, \mathsf{pp}_{\mathsf{TE}}, \mathsf{crs}_{\mathcal{R}}, \mathsf{crs}_{\mathcal{VD}})$
- Parse $\mathsf{sk}^{\mathcal{GM}}$ as $(\mathsf{sk}_{\mathsf{E}}^{\mathcal{GM}}, \mathsf{sk}_{\mathsf{S}}^{\mathcal{GM}})$
- Output $\mathsf{req} \leftarrow \mathsf{S.Sign}_{\mathsf{sk}_{\mathsf{S}}^{\mathcal{GM}}}(\mathsf{m} \| \pi)$

$\underline{\mathsf{JudgeReq}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{req})$
- Parse pp as $(\mathsf{pp}_{\mathsf{E}}, \mathsf{pp}_{\mathsf{S}}, \mathsf{pp}_{\mathsf{TE}}, \mathsf{crs}_{\mathcal{R}}, \mathsf{crs}_{\mathcal{VD}})$
- Parse PK as $((\mathsf{vk}_{\mathsf{S}}^{\mathcal{KI}}), (\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, \mathsf{vk}_{\mathsf{S}}^{\mathcal{GM}}), (\mathsf{pk}_{\mathsf{TE}}^{\mathcal{GU}}))$
- Output $b \leftarrow \mathsf{S.Vrfy}_{\mathsf{vk}_{\mathsf{S}}^{\mathcal{GM}}}(\mathsf{m} \| \pi, \mathsf{req})$

$\underline{\mathsf{GrantDeanon}}\left\langle \mathsf{C}_1(\mathsf{sk}_1^{\mathcal{GU}}), \ldots, \mathsf{C}_n(\mathsf{sk}_n^{\mathcal{GU}}) \right\rangle (\mathsf{pp}, \mathsf{PK}, \mathsf{req}, \mathsf{m}, \pi)$

Upon receiving request $(\mathsf{req}, \mathsf{m}, \pi)$ each party C_l in $\mathcal{GU}$ proceeds as follow:
- Parse pp as $(\mathsf{pp}_{\mathsf{E}}, \mathsf{pp}_{\mathsf{S}}, \mathsf{pp}_{\mathsf{TE}}, \mathsf{crs}_{\mathcal{R}}, \mathsf{crs}_{\mathcal{VD}})$
- Parse PK as $((\mathsf{vk}_{\mathsf{S}}^{\mathcal{KI}}), (\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, \mathsf{vk}_{\mathsf{S}}^{\mathcal{GM}}), (\mathsf{pk}_{\mathsf{TE}}^{\mathcal{GU}}))$
- If $\mathsf{JudgeReq}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{req}) = 0 \vee \mathsf{AuthVrfy}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi) = 0$ it aborts.
- Run protocol $(\mathsf{proof}_{\mathsf{m}}, \mathsf{m} \| c_1) \leftarrow \mathsf{TE.ProveValidDec}\left\langle \mathsf{C}_\ell(\mathsf{sk}_{\mathsf{TE}}^{\ell}) \right\rangle_{\ell=1}^{n}(c_2)$ and output $\mathsf{access} := (\mathsf{proof}_{\mathsf{m}}, \mathsf{m} \| c_1)$

$\underline{\mathsf{Deanon}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{sk}^{\mathcal{GM}})$.
- Parse pp as $(\mathsf{pp}_{\mathsf{E}}, \mathsf{pp}_{\mathsf{S}}, \mathsf{pp}_{\mathsf{TE}}, \mathsf{crs}_{\mathcal{R}}, \mathsf{crs}_{\mathcal{VD}})$
- Parse PK as $((\mathsf{vk}_{\mathsf{S}}^{\mathcal{KI}}), (\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, \mathsf{vk}_{\mathsf{S}}^{\mathcal{GM}}), (\mathsf{pk}_{\mathsf{TE}}^{\mathcal{GU}}))$
- Parse access as $(\mathsf{proof}_{\mathsf{m}}, \mathsf{m} \| c_1)$ If $\mathsf{TE.ValidDec}(\mathsf{m} \| c_1, c_2, \mathsf{proof}_m) = 0$ output $\perp$
- Parse $\mathsf{sk}^{\mathcal{GM}} = (\mathsf{sk}_{\mathsf{E}}^{\mathcal{GM}}, \mathsf{sk}_{\mathsf{S}}^{\mathcal{GM}})$
- $\mathsf{cert}_{\mathrm{ID}} \leftarrow \mathsf{E.Dec}_{\mathsf{sk}_{\mathsf{E}}^{\mathcal{GM}}}(c_1)$
- Parse $\mathsf{cert}_{\mathrm{ID}} = (\mathrm{ID}, \mathsf{vk}, \sigma)$
- If $\mathsf{S.Vrfy}_{\mathsf{vk}_{\mathsf{S}}^{\mathcal{KI}}}(\mathrm{ID} \| \mathsf{vk}, \sigma) = 0$, output $\perp$
- $\mathsf{proof}_{\mathrm{ID}} \leftarrow \mathsf{NIZK}_{\mathcal{VD}}.\mathsf{Prove}\ (\mathsf{crs}_{\mathcal{VD}}, (\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, c_1, \mathsf{cert}_{\mathrm{ID}}), \mathsf{sk}_{\mathsf{E}}^{\mathcal{GM}})$
- Output $(\mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$

$\underline{\mathsf{Judge}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})}$
- Parse pp as $(\mathsf{pp}_{\mathsf{E}}, \mathsf{pp}_{\mathsf{S}}, \mathsf{pp}_{\mathsf{TE}}, \mathsf{crs}_{\mathcal{R}}, \mathsf{crs}_{\mathcal{VD}})$
- Parse PK as $((\mathsf{vk}_{\mathsf{S}}^{\mathcal{KI}}), (\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, \mathsf{vk}_{\mathsf{S}}^{\mathcal{GM}}), (\mathsf{pk}_{\mathsf{TE}}^{\mathcal{GU}}))$
- Parse access as $(\mathsf{proof}_m, \mathsf{m} \| c_1)$
- Parse $\mathsf{cert}_{\mathrm{ID}} = (\mathrm{ID}, \mathsf{vk}, \sigma)$
- If $\mathsf{TE.ValidDec}(\mathsf{m} \| c_1, c_2, \mathsf{proof}_m) = 0$ or $\mathsf{NIZK}_{\mathcal{VD}}.\mathsf{Vrfy}(\mathsf{crs}_{\mathcal{VD}}, (\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, c_1, \mathsf{cert}_{\mathrm{ID}}), \mathsf{proof}_{\mathrm{ID}}) = 0$ output 0
- Output $b \leftarrow \mathsf{S.Vrfy}_{\mathsf{vk}_{\mathsf{S}}^{\mathcal{KI}}}(\mathrm{ID} \| \mathsf{vk}, \sigma)$

Fig. 10. Protocol Π-MAL: De-anonymization Algorithms.

6.1 Preliminaries for Camenisch et al.

Cryptographic Assumptions. The security of this instantiation relies on the modified q-strong Diffie-Hellman assumption [29], the symmetric discrete logarithm assumption and the symmetric external decisional Diffie-Hellman assumption. We give the corresponding definitions for the former two and omit the latter which essentially states that in a pairing group, the decisional Diffie-Hellman assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

Definition 8. *Let $\mathcal{G}$ be a type-3 pairing group generator. The q-MSDH assumption [29] holds with respect to $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$, all $\lambda \in \mathbb{N}$ the probability that $\mathcal{A}\left(\mathsf{gk}, g^a, h^a, h^{ax}, (g^{x^\ell}, h^{x^\ell})_{\ell=0}^{q}\right)$ where $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$, $g \leftarrow \mathbb{G}_1$, $h \leftarrow \mathbb{G}_2$ and $a, x \leftarrow \mathbb{Z}_p$, outputs a tuple $(w, P, \hat{g}^{1/x+w}, \hat{g}^{a/P(x)})$ with $\hat{g} \in \mathbb{G}_1^*$, $P \in \mathbb{Z}_p[X]$ of degree at most q and w such that the polynomials $X + w$ and P are co-prime is negligible in λ.*

Definition 9. *Let $\mathcal{G}$ be a type-3 pairing group generator. The SDL assumption holds with respect to $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$, all $\lambda \in \mathbb{N}$ the probability that $\mathcal{A}(\mathsf{gk}, g, h, g^x, h^x)$ outputs x when $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$, $g \leftarrow \mathbb{G}_1$, $h \leftarrow \mathbb{G}_2$ and $x \leftarrow \mathbb{Z}_p$ is negligible in λ.*

	Communication complexity/size	Time complexity complexity
$\mathcal{KI}/\mathcal{GM}$ generation	-	$\mathcal{O}_\lambda(1)$
Join User	$\mathcal{O}_\lambda(1)$	$\mathcal{O}_\lambda(1)$
$\mathcal{GU}$ generation	$\mathcal{O}_\lambda(n)$ per party	$\mathcal{O}_\lambda(n)$ per party
Proof generation	$\mathcal{O}_\lambda(1)$	$\mathcal{O}_\lambda(\lvert\mathcal{C}\rvert \log \lvert\mathcal{C}\rvert)$
Proof verification	-	$\mathcal{O}_\lambda(1)$
Request	$\mathcal{O}_\lambda(1)$	$\mathcal{O}_\lambda(1)$
Grant	$\mathcal{O}_\lambda(1)$ per party	$\mathcal{O}_\lambda(t)$
De-anonymization	$\mathcal{O}_\lambda(1)$	$\mathcal{O}_\lambda(t)$
Judge	-	$\mathcal{O}_\lambda(t)$

Table 11. Costs for Π-MAL instantiation. λ denotes the security parameter, n the size of the guardians committee, t the desired corruption threshold, and $|\mathcal{C}|$ the number of multiplication gates of the circuit for verifying $\mathcal{R}$. We denote with $\mathcal{O}_\lambda(\cdot)$ asymptotic terms which hide a multiplicative, linear factor λ.

Pointcheval Sanders Signatures. Pointcheval Sanders (PS) signatures [29] are an efficient signature scheme in the bilinear group setting. The structure of PS signatures makes it easy to create a zero knowledge proof of knowledge (ZKPoK) of a verifying message-signature pair. The ZKPoK does not reveal information about the message or the signature. While this can be done for any signature scheme using generic zero knowledge proofs, the ZKPoK of the PS scheme relies on a sigma protocol, which is far more efficient. We next present a high-level description of the PS construction and the ZKPoK.

The PS signature scheme is used to enroll users in the system. The user communicates with the key issuer to enroll itself in the system. Briefly, the user chooses as a private a message m and sends it to the key issuer, who signs it. To explain this more thoroughly, we need to introduce the notation and setting of the PS signature scheme, which uses bilinear pairings. For reference, we describe the PS scheme in Figure 13 and the reader can cross-check our text description below with the figure.

The signature scheme is constructed in a bilinear group, described by $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ and a bilinear map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. All groups have order p and g, h are the generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively. The user chooses as its private key a message m and sends it to the key issuer in $\mathbb{G}_2$. The key issuer signs m. Using the vector (x, y_0, y_1) as the signing key, the key issuer samples a random value a and computes $x + my_0 + ay_1$. The signature is thus the tuple $(a, g, g^{x+my_0+ay_1})$, while the public key is $(X, Y_0, Y_1) = (h^x, h^{y_0}, h^{y_1})$.

To verify the signature $(a, \sigma_1 = g, \sigma_2 = g^{x+my_0+ay_1})$ has been correctly computed, the user checks first $e(\sigma_1, XY_0^m Y_1^a) = e(g, h)^{x+y_0 m+y_1 a}$ and then computes $e(\sigma_2, h)$ and compares the two values. It is easy to see that the values are equal for valid signatures, which ensures completeness.

ZKPoK of verifying PS message-signature. The user can prove without revealing its secret key that it has been enrolled in the group. Using a sigma protocol, the user proves that it knows exponents m, a that satisfy the verification equation $e(g, XY_0^m Y_1^a) = e(\sigma_2, h)$, for given values $h, X, Y_0, Y_1, g', \sigma_2$. This works, but σ_1, σ_2 still leak information about the group elements of the signature. To avoid such information leakage, the prover first *re-randomizes* the signature by computing $(\sigma_1', \sigma_2') = ({\sigma_1}^z, {\sigma_2}^z)$ for a fresh $z \in \mathbb{Z}_q$. The re-randomized signature is valid and only replaces g with g^z. Furthermore, by the DDH assumption in $\mathbb{G}_1$, the new values are pseudorandom and thus reveal no information about σ_1, σ_2. The user publishes these values and uses the mentioned sigma protocol to prove that it knows a verifying message-signature pair and is thus an authorized member of the group. Finally, the sigma protocol can be made non-interactive via the Fiat-Shamir heuristic. The user can then turn the ZKPoK of a verifying PS message-signature pair into a *signature of knowledge* (SoK) of the pair. A SoK means that a user can issue a signature on behalf of any NP statement proving that it knows the witness associated to a statement in the language and that it has signed message m.

*NIZK Protocols used in threshold-*DGS. Above, we described the joining procedure and the procedure through which a user can prove it belongs to the group, and thus give validity to its messages. As we mentioned before,

$\underline{\mathsf{KeyGen}}(gk)$:

- Sample $x, y_0, y_1 \leftarrow \mathbb{Z}_q$ and $h \leftarrow \mathbb{G}_2$. Compute $X := h^x, Y_0 := h^{y_0}, Y_1 := h^{y_1}$.
- Output $(\mathsf{sk}, \mathsf{pk})$ where $\mathsf{sk} := (x, y_0, y_1)$ and $\mathsf{vk} := (X, Y_0, Y_1)$.

$\underline{\mathsf{Sign}}(gk, \mathsf{sk}, m)$:

- Sample $g \leftarrow \mathbb{G}_1$ and $a \leftarrow \mathbb{Z}_p$.
- Output $\sigma := (a, \sigma_1 := g, \sigma_2 := g^{(x+my_0+ay_1))})$.

$\underline{\mathsf{Vrfy}}(\mathsf{vk}, m, \sigma)$: output 1 iff

- Parse $\sigma := (a, \sigma_1, \sigma_2)$.
- Check $e(\sigma_1, XY_0^m Y_1^a) \stackrel{?}{=} e(\sigma_2, h)$

Fig. 12. Pointcheval-Sanders signature scheme.

an important feature of Camenisch et al. construction is that users can be de-anonymized but that the power to perform such a procedure is distributed among many parties, using a Threshold Encryption Scheme.

At high-level, [8] uses a modified version of the Chaum-Pedersen protocol [34] to prove that pairs of elements in $\mathbb{G}_1$ and $\mathbb{G}_T$ have the same discrete logarithm. In other words, these look like standard proofs of knowledge for El Gamal encryption/decryption, but operations are performed under a homomorphism in the target group. For example, consider an El Gamal ciphertext $(C_1, C_2) = (C_1, MC_1^z)$ under public key $Z = h^z$ in group $\mathbb{G}_2$. Let the element $T = e(\Sigma, M)$ be a mapping f of the plaintext M in $\mathbb{G}_T$ such that $f : M \mapsto e(\Sigma, M)$. One can prove that T is the f-mapping of the decryption by proving knowledge of z such that $Z = h^z$ and $T' = e(\Sigma, C_1)^z$ where $T' = e(\Sigma, C_2) \cdot T^{-1}$.

More specifically, we are interested in the following relations regarding El Gamal:

1. Proving correct computation of an encryption (C_1, C_2) of a message M when a homomorphism $f(M)$ is given, namely, for a given Σ, $f(M) = e(\Sigma, M)$
2. Proving correct computation of a homomorphism of a decryption $f(M)$ of a ciphertext (C_1, C_2) namely, for some given Σ, $f(M) = e(\Sigma, M)$.

To go into further detail, we list all the Σ-protocols used. In the following list, $\mathsf{NIZK}_{(\cdot)}$ to denotes a non-interactive zero-knowledge protocol for a language $\mathcal{L}_{(\cdot)}$.

- $\mathsf{NIZK}_{\mathsf{PS}}$: $\mathcal{L}_{\mathsf{PS}} = \left\{\mathsf{pk} \mid \exists (m, \sigma) \text{ s.t. } \mathsf{Vrfy}_{\mathsf{pk}}(\mathsf{gk}, m, \sigma) = 1\right\}$
- $\mathsf{NIZK}_{\mathsf{DDH}}$: $\mathcal{L}_{\mathsf{DDH}} = \{a, b, c, d \in \mathbb{G}_1 \mid \exists w \text{ s.t. } b = a^w \wedge d = c^w\}$
- $\mathsf{NIZK}_{\mathsf{T}}$: $\mathcal{L}_{\mathsf{T}} = \{a, b \in \mathbb{G}_2, c, d \in \mathbb{G}_T \mid \exists w \text{ s.t. } b = a^w \wedge d = c^w\}$
- $\mathsf{NIZK}_{\mathsf{enc}}$: $\mathcal{L}_{\mathsf{enc}} =$

$$\left\{ \begin{array}{c|c} \Sigma \in \mathbb{G}_1, & \exists r \text{ s.t.} \\ H, Z, C_1, C_2 \in \mathbb{G}_2, & C_1 = H^r \wedge \\ T \in \mathbb{G}_T & e(\Sigma, C_2) \cdot T^{-1} = e(\Sigma, Z)^r \end{array} \right\}$$

- $\mathsf{NIZK}_{\mathsf{Dec}}$: $\mathcal{L}_{\mathsf{Dec}} = \left\{\Sigma \in \mathbb{G}_1, H, Z, C_1, C_2 \in \mathbb{G}_2, T \in \mathbb{G}_T \mid \exists z \text{ s.t. } Z = H^z \wedge e(\Sigma, C_2) \cdot T^{-1} = e(\Sigma, C_1)^z\right\}$

Note the last two proofs are simple instantiations of $\mathsf{NIZK}_{\mathsf{T}}$, specifically,

- For $\mathsf{NIZK}_{\mathsf{Enc}}$ we can set $a = H, b = C_1, c = e(\Sigma, Z), d = e(\Sigma, C_2)T^{-1}$ and use $\mathsf{NIZK}_{\mathsf{T}}$.
- For $\mathsf{NIZK}_{\mathsf{Dec}}$ we can set $a = H, b = C_1, c = e(\Sigma, C_1), d = e(\Sigma, C_2)T^{-1}$ and use $\mathsf{NIZK}_{\mathsf{T}}$.

Now, one can prove knowledge of field elements m, a such that $T_m^m T_a^a = T$ where $T_m = e(\sigma_1, Y_0), T_a = e(\sigma_1, Y_1)$ and $T = e(\sigma_2, h)e(\sigma_1, X)^{-1}$.

6.2 Overview of Camenisch et al. Construction

We next present a final overview of the construction due to Camenisch et al. [8], which we refer to as DGS. As mentioned before, [8] focuses on solving the trust issue that arises in group signature constructions, namely, the power of the party who enrolls users in a group and the one the entity that can de-anonymize users has. For both cases, [8] distribute these roles by using threshold cryptography techniques and solve both issues simultaneously: their scheme has two different committees – one for each task – and a number of parties in each can be corrupted without compromising the security of the scheme.

We briefly discuss how the schemes presented above are used in the algorithms for enrolling users in the group, signing a message (which corresponds to generating a membership authorization in our general setting), and de-anonymization.

To join the group, a user U generates its secret key sk and asks the key issuer[10] to blindly signing it, and the latter generates a PS signature (a, σ_1, σ_2) on sk. that U sampled.

To generate a proof of membership, U creates a signature of knowledge (SoK) of a valid message/PS signature pair under the key issuer's verification key, as explained above. The SoK guarantees no information is revealed about the message/PS signature and therefore about the user secret key. Furthermore, only group members can create proofs of membership: this follows by the fact that signing a message involves issuing a signature of knowledge on this message which in turn implies knowing a verifying PS message-signature pair issued by the key issuer. .

In the follow, we explain how the de-anonymization procedure for a message m works, namely the ability of a designated party to identify the signer of a specific message. Note that the designated party in this case is a committee. The user secret-shares the value Y_0^{sk} with the committee, where $Y_0 = h^{y_0}$ is part of the PS signature public key as described in the previous section. The de-anonymization procedure requires the committee, to recover $Y_0^{\mathsf{sk}_i}$ for each user U_i in the system and perform the test of the verification equation with m to see which specific user signed it. If the openers compute $Y_0^{\mathsf{sk}_i}$ for every user (as required to do de-anonymization), say to de-anonymize one specific malicious message, they can individually use these values to de-anonymize every message. This is prevented by requiring that the committee never recovers $Y_0^{\mathsf{sk}_i}$. They instead compute the values $e(\sigma_1', Y_0^{\mathsf{sk}_i})$. Using these values, they can still do the check against a signature $(a, \sigma_1', \sigma_2')$. However, these values are useless for other messages since they are "tied" to the value σ_1'. Verifiable secret sharing techniques and sigma protocols are used to perform the above actions verifiably, both from the side of the users as well as the openers.

On the efficiency of the construction. The SoK based on Pointcheval-Sander's signature scheme uses a sigma protocol as proof of knowledge and, because sigma protocols are efficient, the resulting membership proofs in this construction are small and signing requires only lightweight computation from users. In contrast, opening signatures is expensive: each opener needs to do work linear in the size of the group itself. However, [8] assumes that de-anonymization is far rarer than signing, so this trade-off is acceptable. Furthermore, the latter task can afford to be performed using stronger hardware, while for many real-world applications one would want proving membership to be lightweight and efficient.

Modifications for MAL. The main goal of our MAL framework is to hold the manager accountable, and thus it is crucial to separate the tasks of de-anonymizing and granting for de-anonymization. The construction of Camenisch et al. can be easily modified to fit this framework by using a more sophisticated access structure -namely a policy for which a subset of parties can reconstruct a shared secret- for secret sharing the value Y_0^{sk} for each user. . Specifically, we require that a shared secret can be reconstructed if at least t out of n parties in the committee (guardians) *and* a special party (group manager) collaborate, and any other subset of parties learns no information about it. This is achieved as follows: first, each user U samples, after sampling its secret key sk, a uniform element sk_1 and sets $\mathsf{sk}_2 = \mathsf{sk} - \mathsf{sk}_1$, then gives sk_2 to the group manager and it uses a standard (t, n)-secret sharing scheme to share sk_1 between the guardians (as done in [8]).

[10] While in [8] it is possible to distribute the role of the key issuer, this is out of the scope of our work, so we will only assume a single key issuer. However, our modified construction can be adapted straightforwardly for such a feature.

$\underline{\mathsf{Setup}}(1^\lambda, t, n)$

- $\mathsf{gk} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, h, e) \leftarrow \mathcal{G}(1^\lambda)$
- $\mathsf{pp} := (\mathsf{gk}, n, t)$

$\underline{\mathsf{KeyGenIssuer}}(\mathsf{pp})$
$\mathcal{KI}$ generates a key pair for PS signature scheme in $\mathbb{G}_2$.

- Sample $x, y_0, y_1 \leftarrow \mathbb{Z}_q$. Compute $X := h^x, Y_0 := h^{y_0}, Y_1 := h^{y_1}$.
- Output $(\mathsf{sk}^{\mathcal{KI}}, \mathsf{pk}^{\mathcal{KI}})$ where $\mathsf{sk}^{\mathcal{KI}} = (x, y_0, y_1)$ and $\mathsf{pk}^{\mathcal{KI}} = (X, Y_0, Y_1)$.

$\underline{\mathsf{KeyGenManager}}(\mathsf{pp})$
$\mathcal{GM}$ generates an El-Gamal key pair on $\mathbb{G}_2$.

- Sample $z \leftarrow \mathbb{Z}_q$. Compute $Z := h^z$.
- Output $(\mathsf{sk}^{\mathcal{GM}}, \mathsf{pk}^{\mathcal{GM}}) = (z, Z)$

$\underline{\mathsf{KeyGenGu}}\ \langle \mathsf{C}_\ell(\cdot)\rangle_{\mathsf{C}_\ell \in \mathcal{GU}}\ (\mathsf{pp})$.
Each guardian C_ℓ generates an El-Gamal key pair on $\mathbb{G}_2$.

- Sample $z_\ell \leftarrow \mathbb{Z}_q$. Compute $Z_\ell := h^{z_\ell}$.
- Output $(\mathsf{sk}^{\mathsf{C}_\ell}, \mathsf{pk}^{\mathsf{C}_\ell}) = (z_\ell, Z_\ell)$

Fig. 13. Parameter setup and Key Generation Protocols and Algorithms. We assume that all parties have as well a signature key pair for authenticated communication, which we omit in the description of the protocol.

Now, upon a valid de-anonymization request, for each user U_i the guardians reconstruct the second part of the secret key $\mathsf{sk}_{i,2}$ by collectively computing $e(\sigma_1', Y^{\mathsf{sk}_{i,2}})$ as in the Camenisch et al. case, and they send these values as the "authorization" to the manager. Then the manager can compute on its own $e(\sigma_1', Y^{\mathsf{sk}_i})$ by using its share $Y^{\mathsf{sk}_{i,1}}$ and perform the final de-anonymization by checking for which value the PS signature associated with the message verifies.

Intuitively, neither the guardians nor the manager alone can reconstruct $e(\sigma_1', Y_0^{\mathsf{sk}})$ and so de-anonymization only happens when an appropriate subset of guardians *and* the group managercollaborates as required by MAL. Also, because group manager is the only party that reconstructs the above value, it is the only party that learns the identity of an issuer of a message. Finally, to ensure manager accountability, the guardians only grant de-anonymization to the group manager if the latter first signs a request; this request proves that the group manager indeed intended to de-anonymize some specific message.

For efficiency, from the user's perspective (enrolling, signing and verifying openings) nothing changes compared to the protocol by Camenisch et al. The situation is like de-anonymization with one change: while in the original DGS, the openers could stop "brute forcing" users when a signer is identified, in the modified version the guardians never learn the signer and so they always need to brute force all the users.

The full description of the scheme is presented in Fig. 13,14,15,16,17.

6.3 Security

We next argue about security of the construction. The proofs are based on the proofs of [8] and the fact that the modified way of sharing the user secret preserves privacy for the access structure of interest. Completeness (Def. 2) holds by inspection.

Unforgeability (Def. 3) holds because a verifying signature of knowledge implies that its producer either (1) knows a statement/witness pair – in this case a PS signature on a message – which contradicts unforgeability of PS signature scheme or (2) manages to produce a verifying PoK without the possession of valid statement/witness pair, thus violating knowledge soundness of $\mathsf{NIZK}_{\mathsf{PS}}$.

For user non-frameability (Def. 4), managing to "blame" an honest user for a message/signature pair means that an adversary either managed to give a valid signature on her behalf (like unforgeability) or cheated in the opening phase. The latter cannot happen because of the verifiable way decryption happens using $\mathsf{NIZK}_{\mathsf{Dec}}$. Similarly, for user accountability (Def. 5) an adversary producing a verifying message/signature pair that fails to open must either create a PS signature on its own for a non-existent user or break knowledge soundness of $\mathsf{NIZK}_{\mathsf{PS}}$. For manager accountability, we reduce to a simple case of unforgeability; it is enough to note that the manager must sign the de-anonymization requests.

The proof of manager non-frameability (Def. 6) naturally goes as in Section 5. Finally, in each of the two cases of anonymity the adversary does not have enough shares to de-anonymize. Further, the shares are encrypted, so it is infeasible for the adversary to use them under indistinguishability of El Gamal in $\mathbb{G}_2$. The proof itself contains a rerandomized signature σ_1', σ_2' and a proof for $\mathsf{NIZK}_{\mathsf{PS}}$. $\mathsf{NIZK}_{\mathsf{PS}}$ satisfies zero knowledge, and therefore reveals no information. Similarly, the re-randomized part is pseudorandom, so neither does it reveal any information. More specifically, since $\sigma_1, \sigma_2, \sigma_1' = \sigma_1^z, \sigma_2' = \sigma_1^z$ is a DDH tuple, and no PPT distinguisher can distinguish a DDH tuple under the DDH assumption in $\mathbb{G}_1$.

We next present the theorems that capture the security property of the construction and give some intuition for the proofs. The full proofs can be found in App. C.

Theorem 6. *In the random oracle model, DGS-MAL achieves unforgeability (Def. 3) under n_U-MSDH and the SDL assumption, where n_U is an upper bound on the number of users.*

Intuition: $\mathcal{A}$ wins Exp-Unforge if it can provide a pair (m, π) such that $\mathsf{AuthVrfy}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi) = 1$ and π is not the output of a query on OMemberAuth. Thus, $\mathcal{A}$ (1) either creates a certificate on its own, which means it forges a PS signature, or (2) manages to create a valid signature of knowledge, which means it can break knowledge soundness of $\mathsf{NIZK}_{\mathsf{PS}}$ or learn sk of one honest user, which is encoded as a discrete logarithm.

Theorem 7. *In the random oracle model, construction DGS-MAL achieves user non-frameability (Def. 4) under the SDL assumption.*

Intuition: If $\mathcal{A}$ wins Exp-UserNonFrame, it either wins unforgeability or fake the decryption of the share of the user, which means the adversary breaks soundness of $\mathsf{NIZK}_{\mathsf{Dec}}$.

Theorem 8. *In the random oracle model, construction DGS-MAL achieves User Accountability (Def.5) under the $q_{\mathcal{H}}$-MSDH, where $q_{\mathcal{H}}$ is the number of queries to $\mathcal{H}$ made by the adversary.*

Intuition: An adversary $\mathcal{A}$ wins if she provides a pair (m, π) that, upon going through the de-anonymization process, does not open to an ID belonging to any user. This means $\mathcal{A}$ should be able to (i) forge PS message signature pair on the key of $\mathcal{KI}$, or (ii) create a proof of membership without holding any valid certificate, forging a signature of knowledge.

Theorem 9. *Construction DGS-MAL achieves Manager Non-Frameability (Def. 6).*

Intuition. The proof goes exactly as the one for the general instantiation and thus we omit it.

Theorem 10. *In the random oracle model, construction DGS-MAL achieves User Anonymity (Def. 7) under the ddh assumption in $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Intuition: An adversary that breaks anonymity manages to distinguish between membership permissions. The proof of knowledge used to create the latter does not reveal any information about the signature, so any adversary that succeeds in distinguishing should gain some information from the randomized values σ_1, σ_2. By the DDH assumption, these are indistinguishable from random, but they are correlated with the shares given during issuing. However, an adversary owing few shares (either all guardians' shares or a minority of them and the manager's share) has no information about the secret.

6.4 Evolving Committees

The security properties introduced in this work require the participation of the committee of guardians. However, in real life a committee of parties may only be available short-term and may also become corrupted over time. One solution is to allow guardians to change via the dynamic proactive secret sharing scheme (DPSS) by Goyal et al. [10] to our setting. DPSS allows a set of n parties, who hold n shares of a secret, to hand-off the secret to another set of n parties. The n shares of the secret in our setting are those corresponding to the secret key associated to $\mathsf{pk}_{\mathsf{TE}}^{\mathcal{GU}}$. The public and secret key of the guardians remain the same, but the shares of the secret key are updated. Users always sign their messages with the same public key and the changing of committee does not affect them.

The DPSS protocol [10] consists of *setup*, *hand-off*, and *reconstruction* phases. In our instantiations (Sections 5, 6) the three phases for DPSS are as follows: (1) In the *setup*, an initial committee produces ElGamal distributed keys. We denote the shared secret s and the public encryption key g^s; (2) in the *hand-off* the old committee transfers the secret to the new committee, and we present this phase in Fig. 18; (3) rather than reconstruction, in our setting, the new committee work together to grant de-anonymizations. The current set of guardians removes the outer layer of decryption using the shared secret key s in a standard execution of verifiable decryption of ElGamal.

The system tolerates adaptive corruptions as soon as the adversary is not able to corrupt $t+1$ parties in the lifetime of one committee. If this is the case, namely, the adversary is capable of corrupting at most t parties from each committee, it learns nothing about the secret (*secrecy*), nor can prevent parties from reconstructing the secret (*robustness*).

The hand-off. We denote by $\mathsf{C}_\ell^{(i)}$ member ℓ of committee i, where each member holds a secret of a value $s \in \mathbb{Z}_p$. Concretely, the secret shares are created by Shamir secret sharing: the share s_ℓ of party ℓ is $p(\ell)$, where $p(X)$ is a random polynomial of degree t s.t. $p(0) = s$. We denote $[s]_t$ the secret shares of s and $[g^s]_t$ the same values but in the exponent of a group generator g used for the ElGamal cryptosystem. Essentially, including these values is the only modification we do in [10] to account for the different case of threshold decryption.

Below, we shortly describe the hand-off phase of [10], i.e., the procedure through which committee i and committee $i+1$ exchange and update their secret keys; parties of the old committee $\mathcal{GU}^{(i)}$ hold a secret sharing of $[s]_t$ which they want to "hand-off" to parties in $\mathcal{GU}^{(i+1)}$. The procedure consists of two sub-phases:

- In the *preparation phase,* the new committee prepares two independent degree-t sharings of a random value, denoted by $[r]_t$ and $[\tilde{r}]_t$ respectively, such that $r = \tilde{r}$, The old committee holds $[r]_t$ and the new committee $[\tilde{r}]_t$.
- In the *refresh phase*, committee i computes the sharing $[s-r]_t = [s]_t - [r]_t$ and reconstructs the secret $s - r$. Members of committee $i+1$ then calculate the secret sharing of $[\tilde{s}]_t = [\tilde{r}]_t + (s-r)$, which is an independent sharing of s, namely $[\tilde{s}]_t$.

After the hand-off phase is successfully completed, each member of the old committee $\mathsf{C}^{(i)}$ erases the old shares $[s]_d$ to avoid future corruptions. To ensure that the shares exchanged between parties are well-formed and to allow honest parties to identify misbehaving ones (meaning they cannot create malformed shares), the protocol relies on a polynomial commitment scheme. Finally, to achieve verifiable decryption the protocol makes use of a nizk argument of knowledge. For simplicity, we omit the above descriptions in the presentation of the protocol. Our protocol closely follows the one introduced in [10] and we present it in Fig. 18.

Evolving Committees in the instantiation of Section 6. When instantiating our framework with the signature scheme by Camenisch et al., we have that each user generates a secret key and distributes its shares with the guardians (and manager). Namely, the guardians store not one but multiple secret shares, one for each user in the system. In this case, the protocol of Goyal et al. can easily be adapted, as presented in [10]. Basically, parties in $\mathcal{GU}^{(i+1)}$ create two independent secrets $(r_i, \tilde{r}_i)$ for each user and batch them and the shares $([r_i]_t, [\tilde{r}_i]_t)$ through a random linear combination that is also used by the parties of $\mathcal{GU}^{(i)}$ with $\{[s_i]_t\}$ to reconstruct $\{[s_i - r_i]_t\}$. As a result, the amortized cost of communication per pair is $O(n)$ elements.

Security Overview. The hand-off phase is particularly challenging, since an adversary can corrupt t parties in the old committee and t parties in the new committee. However, after the hand-off phase, guardians from the old committee have the instruction to delete their shares and thus, while corrupting $t+1$ guardians from any previous committee is enough for the adversary to recover the secret key, it can happen with only a small probability.

In security for the semi-honest case, parties distribute consistent shares. An adversary controls t parties of the old committee and t parties of the new committee. If the two coupled sharings are random, then an adversary having t shares of each sees only the share s masked with a random value r as $s - r$, and thus cannot learn anything about s.

In [10], the approach used is to have each member of the new committee create an individual random coupled share $[r_j]_t, [\tilde{r_j}]_t$ and then all parties collectively compute the coupled sharing $([r]_t, [\tilde{r}]_t) = ([\sum r_j]_t, [\sum r_j]_t)$. Since at least one party is honest, say party ℓ, the value r_ℓ is uniformly distributed, and thus, so is r. To achieve security in the malicious case, the authors in [10] use the polynomial commitment scheme of Kate et. al. [37] to ensure that the values shared by each party are consistent, namely, that party ℓ receives the share $p(\ell)$ for a committed polynomial p of degree t. They also take advantage of the linear-homomorphic properties of [37] to improve efficiency and create efficient accuse-response protocols to identify misbehaving parties. We refer the reader to [10, Sec. 4, 5] for the details.

References

1. Luis von Ahn, Andrew Bortz, Nicholas J. Hopper, and Kevin O'Neill. Selectively traceable anonymity. In George Danezis and Philippe Golle, editors, *PET 2006*, volume 4258 of *LNCS*, pages 208–222, Cambridge, UK, June 28–30, 2006. Springer, Heidelberg, Germany.
2. Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel J. Weitzner. Practical accountability of secret processes. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 657–674, Baltimore, MD, USA, August 15–17, 2018. USENIX Association.
3. Stefan Köpsell, Rolf Wendolsky, and Hannes Federrath. Revocable anonymity. In *International Conference on Emerging Trends in Information and Communication Security*, pages 206–220. Springer, 2006.
4. Benoît Libert, Fabrice Mouhartem, and Khoa Nguyen. A lattice-based group signature scheme with message-dependent opening. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 137–155, Guildford, UK, June 19–22, 2016. Springer, Heidelberg, Germany.
5. Benoît Libert and Marc Joye. Group signatures with message-dependent opening in the standard model. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 286–306, San Francisco, CA, USA, February 25–28, 2014. Springer, Heidelberg, Germany.
6. Yusuke Sakai, Keita Emura, Goichiro Hanaoka, Yutaka Kawai, Takahiro Matsuda, and Kazumasa Omote. Group signatures with message-dependent opening. In Michel Abdalla and Tanja Lange, editors, *PAIRING 2012*, volume 7708 of *LNCS*, pages 270–294, Cologne, Germany, May 16–18, 2013. Springer, Heidelberg, Germany.
7. Joris Claessens, Claudia Diaz, Caroline Goemans, Jos Dumortier, Bart Preneel, and Joos Vandewalle. Revocable anonymous access to the internet? *Internet Research*, 2003.
8. Jan Camenisch, Manu Drijvers, Anja Lehmann, Gregory Neven, and Patrick Towa. Short threshold dynamic group signatures. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 401–423, Amalfi, Italy, September 14–16, 2020. Springer, Heidelberg, Germany.
9. Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 339–352, Santa Barbara, CA, USA, August 27–31, 1995. Springer, Heidelberg, Germany.
10. Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. Cryptology ePrint Archive, Report 2020/504, 2020. `https://eprint.iacr.org/2020/504`.
11. Claudia Diaz and Bart Preneel. Accountable anonymous communication. In *Security, Privacy, and Trust in Modern Data Management*, pages 239–253. Springer, 2007.
12. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

13. Dan Boneh, Rosario Gennaro, Steven Goldfeder, and Sam Kim. A lattice-based universal thresholdizer for cryptographic systems. Cryptology ePrint Archive, Report 2017/251, 2017. http://eprint.iacr.org/2017/251.
14. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130, New York, NY, USA, October 17–19, 1999. IEEE Computer Society Press.
15. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017. http://eprint.iacr.org/2017/454.
16. Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
17. Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In *Theory of Cryptography Conference*, pages 260–290. Springer, 2020.
18. David A. Schultz, Barbara Liskov, and Moses Liskov. Mobile proactive secret sharing. In Rida A. Bazzi and Boaz Patt-Shamir, editors, *27th ACM PODC*, page 458, Toronto, Ontario, Canada, August 18–21, 2008. ACM.
19. Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 340–350, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.
20. Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Heidelberg, Germany.
21. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
22. George Danezis and Len Sassaman. How to bypass two anonymity revocation schemes. In Nikita Borisov and Ian Goldberg, editors, *PETS 2008*, volume 5134 of *LNCS*, pages 187–201, Leuven, Belgium, July 23–25, 2008. Springer, Heidelberg, Germany.
23. Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.
24. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.
25. Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.
26. Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.
27. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.
28. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.
29. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126, San Francisco, CA, USA, February 29 – March 4, 2016. Springer, Heidelberg, Germany.
30. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.
31. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany.
32. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.
33. Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th FOCS*, pages 427–437, Los Angeles, CA, USA, October 12–14, 1987. IEEE Computer Society Press.
34. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany.

35. Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.
36. Wouter Penard and Tim van Werkhoven. On the secure hash algorithm family. *Cryptography in Context*, pages 1–18, 2008.
37. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany.

A Cryptographic Primitives

A.1 Threshold Encryption Scheme

A (t, n)-Threshold Encryption Scheme TE is a tuple of algorithms and protocols ($\mathsf{TE.Setup}$, $\mathsf{TE.KeyGen}$, $\mathsf{TE.Enc}$, $\mathsf{TE.ProveValidDec}$, $\mathsf{TE.ValidDec}$) executed by n parties:

- $\mathsf{pp_{TE}} \leftarrow \mathsf{TE.Setup}(n, t, 1^\lambda)$: On input the number of parties n, a threshold t, and the security parameter 1^λ, it outputs parameters for the scheme $\mathsf{pp_{TE}}$.
- $(\mathsf{pk_{TE}}, (\mathsf{sk}^i_{\mathsf{TE}})_{i=1}^n) \leftarrow \mathsf{TE.KeyGen}\langle U_i(\cdot)\rangle_{i=1}^n(\mathsf{pp_{TE}})$: An n-party protocol with public input the parameters $\mathsf{pp_{TE}}$ and no secret input. The common output is $\mathsf{pk_{TE}}$ and party i receives a secret output $\mathsf{sk}^i_{\mathsf{TE}}$.
- $c \leftarrow \mathsf{TE.Enc}_{\mathsf{pk_{TE}}}(\mathsf{m})$. On input the public key $\mathsf{pk_{TE}}$ and a message m, it outputs a ciphertext c.
- $(\mathsf{proof_m}, \mathsf{m}) \leftarrow \mathsf{TE.ProveValidDec}\langle U_i(\mathsf{sk}^i_{\mathsf{TE}})\rangle_{i=1}^n(c)$: on input a ciphertext c, the n-party protocol produces a decryption and a publicly verifiable proof that the decryption was done correctly
- $b \leftarrow \mathsf{TE.ValidDec}_{\mathsf{pk_{TE}}}(\mathsf{m}, c, \mathsf{proof_m})$: On input a ciphertext c, a message m, and a proof $\mathsf{proof_m}$, it outputs a bit b indicating whether m is the correct decryption of c.

We require a threshold encryption scheme to satisfy correctness, computational soundness, indistinguishability of encryptions, and decryption simulatability. We omit correctness which states that $t + 1$ honest can always decrypt.

Soundness states that it should be infeasible to produce fake proofs of correct decryption.

Definition 10. *[Computational Soundness] For all PPT $\mathcal{A}$, all $\lambda, n, t \in \mathbb{N}$, with $t \leq n$,*

$$\Pr\left[\mathsf{Soundness}_{\mathsf{TE},\mathcal{A}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

Indistinguishability of encryptions captures that no adversary can distinguish ciphertexts even when he is able to control up to t parties, selected adaptively during key generation and decryption.

Definition 11. *[(t, n)-Indistinguishability of Encryption] For all PPT adversaries $\mathcal{A}$,*

$$\Pr\left[(t,n)\text{-}\mathsf{Ind}_{\mathsf{TE},\mathcal{A}}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Finally, decryption simulatability means that given a message and its encryption, it is possible to simulate the proof that the decryption was properly computed.

Definition 12 (t-Decryption Simulatability). *A Threshold Encryption Scheme has t-Decryption Simulatability if there exists a PPT algorithm $\mathcal{S}$, such that for all PPT $\mathcal{A}$, messages m, randomnesses r, and sets $S_1 \subseteq S_2 \subseteq [n]$ with $|S_2| \leq t$, the following distributions are perfectly indistinguishable:*

$$\left\{ \begin{array}{c} (\mathsf{proof_m}, \mathsf{m}) \leftarrow \mathsf{TE.ProveValidDec} \\ (\mathcal{A}(\mathsf{st}_\mathcal{A}))\langle U_i(\mathsf{sk}^i_{\mathsf{TE}})\rangle_{U_i \in [n]\setminus S_2}(c) \end{array} \middle| \begin{array}{r} \mathsf{pp_{TE}} \leftarrow \mathsf{TE.Setup}(n, t, 1^\lambda); \\ \left(\mathsf{pk_{TE}}, \mathsf{st}_\mathcal{A}, (\mathsf{sk}^i_{\mathsf{TE}})_{U_i \in [n]\setminus S_1}\right) \leftarrow \mathsf{TE.KeyGen}(\mathcal{A}(\cdot))\langle U_i(\cdot)\rangle_{U_i \in [n]\setminus S_1}(\mathsf{pp_{TE}}) \\ c \leftarrow \mathsf{TE.Enc}_{\mathsf{pk_{TE}}}(\mathsf{m}; r) \end{array} \right\}$$

$$\left\{ \begin{array}{c} (\mathsf{proof}_{\mathrm{sim}}, \mathsf{m}) \leftarrow \mathsf{TE.ProveValidDec} \\ (\mathcal{A}(\mathsf{st}_\mathcal{A}), \mathcal{S}(\mathsf{m}, r))(c) \end{array} \middle| \begin{array}{r} \mathsf{pp_{TE}} \leftarrow \mathsf{TE.Setup}(n, t, 1^\lambda); \\ \left(\mathsf{pk_{TE}}, \mathsf{st}_\mathcal{A}, (\mathsf{sk}^i_{\mathsf{TE}})_{U_i \in [n]\setminus S_1}\right) \leftarrow \mathsf{TE.KeyGen}(\mathcal{A}(\cdot))\langle U_i(\cdot)\rangle_{U_i \in [n]\setminus S_1}(\mathsf{pp_{TE}}) \\ c \leftarrow \mathsf{TE.Enc}_{\mathsf{pk_{TE}}}(\mathsf{m}; r) \end{array} \right\}$$

A.2 Non-Interactive Zero-Knowledge Argument

Let $\mathcal{R}$ be an NP relation, with pairs $(u, \mathsf{w}) \in \mathcal{R}$ where u is known as the *statement* and w is known as the *witness*.

Definition 13 (Non-Interactive Zero-Knowledge). *A Non-Interactive Zero-Knowledge Argument of Knowledge* $\mathsf{NIZK}_{\mathcal{R}}$ *for* $\mathcal{R}$*, is a tuple of algorithms* $(\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}, \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}, \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Vrfy}, \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Sim})$ *such that:*

- $(\mathsf{crs}, \tau) \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(\lambda)$*: on input a security parameter* λ*, it outputs a common reference string* crs *and a simulation trapdoor* τ*.*
- $\mathsf{proof} \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}(\mathsf{crs}, u, \mathsf{w})$*: on input* crs *and a pair* $(u, w) \in \mathcal{R}$*, it outputs a proof* proof*.*
- $b \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Vrfy}(\mathsf{crs}, u, \mathsf{proof})$*: on input* crs*, a statement* u *and an argument* proof*, it returns a bit* b *indicating acceptance (1) or rejection (0).*
- $\mathsf{proof}_{\mathrm{sim}} \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Sim}(\tau, u)$ *: on input the simulation trapdoor* τ *and a statement* u*, it returns a simulated proof* $\mathsf{proof}_{\mathrm{sim}}$*.*

for which the following properties hold:

Definition 14 (Completeness). *For all* $(u, w) \in \mathcal{R}$*, we have that, if we execute* $(\sigma, \tau) \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(\lambda)$ *and* $\mathsf{proof} \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}(\sigma, u, w)$*, then* $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Vrfy}(\sigma, u, \mathsf{proof}) = 1$ *with probability* 1*.*

Definition 15 (Knowledge-Soundness). *For all PPT adversaries* $\mathcal{A}$*, there exists a PPT extractor* $\mathcal{X}_{\mathcal{A}}$ *and a negligible function* μ*, such that,*

$$\Pr\left[\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Vrfy}(\sigma, u, \mathsf{proof}) = 1 \wedge (u, w) \notin \mathcal{R} \,\middle|\, \begin{array}{r}(\mathsf{crs}, \tau) \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(\lambda) \\ (u, \mathsf{proof}) \leftarrow \mathcal{A}(\mathsf{crs}; r), \\ w \leftarrow \mathcal{X}_{\mathcal{A}}(\mathsf{crs}; r)\end{array}\right] \leq \mu(\lambda)$$

Definition 16. *(Zero-Knowledge.) For all adversaries* $\mathcal{A}$*, there exists a negligible function* μ*, such that,*

$$\Pr\left[b' = b \middle| (\mathsf{crs}, \tau) \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(\lambda) \wedge b \leftarrow \{0,1\} \ \wedge b' \leftarrow \mathcal{A}^{\mathcal{O}_b}(\mathsf{crs})\right] \leq \frac{1}{2} + \mu(\lambda)$$

where

$$\mathcal{O}_0(u, w) \leftarrow \begin{cases} \perp & (u, w) \notin \mathcal{R} \\ \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}(\mathsf{crs}, u, w) & \textit{otherwise} \end{cases}, \qquad \mathcal{O}_1(u, w) \leftarrow \begin{cases} \perp & (u, w) \notin \mathcal{R} \\ \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Sim}(\tau, u) & \textit{otherwise} \end{cases}.$$

Additionally, $\mathsf{NIZK}_{\mathcal{R}}$ can satisfy the following property:

Definition 17 (Simulation-extractability). *For all PPT adversaries* $\mathcal{A}$*, there exists a PPT extractor* $\mathcal{X}_{\mathcal{A}}$ *and a negligible function* μ*, such that,*

$$\Pr\left[\begin{array}{l}\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Vrfy}(\sigma, u, \mathsf{proof}) = 1 \\ \wedge (u, w) \notin \mathcal{R} \wedge (u, \mathsf{proof}) \notin Q\end{array}\middle| \begin{array}{r}(\mathsf{crs}, \tau) \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(\lambda) \\ (u, \mathsf{proof}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{crs}; r), w \leftarrow \mathcal{X}_{\mathcal{A}}(\mathsf{crs}; r)\end{array}\right] \leq \mu(\lambda)$$

where $\mathcal{O}(\cdot)$ *is an oracle to* $\mathsf{NIZK.Sim}$ $(\tau, \cdot)$ *and* Q *is the set of all oracle query-answer pairs.*

When a NIZK argument satisfies Simulation-extractability, we will call it a SNARK, as in [28], and denote $\mathsf{snark}_{\mathcal{R}} = \big(\mathsf{snark}_{\mathcal{R}}.\mathsf{Setup}, \mathsf{snark}_{\mathcal{R}}.\mathsf{Prove}, \mathsf{snark}_{\mathcal{R}}.\mathsf{Vrfy}, \mathsf{snark}_{\mathcal{R}}.\mathsf{Sim}\big)$.

B Security of the generic construction

B.1 Proof of Thm. 1

Proof. We prove unforgeability via hybrid arguments and a reduction to One Wayness.
$\mathsf{Hyb}_0^{\mathcal{A}} \to \mathsf{Hyb}_1^{\mathcal{A}}$**:**. We denote the real unforgeability game (Def. 2), instantiated with Π-MAL as $\mathsf{Hyb}_0^{\mathcal{A}}$. Then $\mathsf{Hyb}_1^{\mathcal{A}}$ is the same as $\mathsf{Hyb}_0^{\mathcal{A}}$ but when simulating the $\mathsf{OMemberAuth}$ oracle, proof is replaced with $\mathsf{proof}_{\mathsf{Sim}} \leftarrow \mathsf{snark}_{\mathcal{R}}.\mathsf{Sim}(\tau_{\mathcal{R}}, (\mathsf{m}, c_2, \mathsf{PK}))$.

Suppose $\mathcal{A}$ is such that $\Pr\left[\mathsf{Hyb}_0^{\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hyb}_1^{\mathcal{A}}(1^\lambda) = 1\right] \geq p(\lambda)$, for a non-negligible function p. Note that the only difference in the two games is how the snark's proofs are constructed. Thus, distinguishing between the two games, directly implies distinguishing honestly generated versus simulated proofs with non-negligible probability, which contradicts the zero knowledge property of the snark.
$\mathsf{Hyb}_1^{\mathcal{A}} \to \mathsf{Hyb}_2^{\mathcal{A}}$**:** $\mathsf{Hyb}_2^{\mathcal{A}}$ is the same as $\mathsf{Hyb}_1^{\mathcal{A}}$ but upon receiving forgery (m, π) from $\mathcal{A}$, parse π as (c_2, proof), invoke the snark extractor $\mathsf{snark}_{\mathcal{R}}.\mathcal{X}(\mathsf{crs}_{\mathcal{R}})$ and output a witness $(r_2, c_1, r_1, \mathsf{cert}, \mathsf{sk})$. Abort if $((\mathsf{m}, c_2, \mathsf{PK}, \mathsf{proof}), (r_2, c_1, r_1, \mathsf{cert}, \mathsf{sk})) \notin \mathcal{R}$.

This games are indistinguishable conditioned on the extractor outputting a valid witness. Suppose this is not the case and all PPT algorithms $\mathsf{snark}_{\mathcal{R}}.\mathcal{X}_{\mathcal{A}}$ fail, with non-negligible probability, to output $w = (r_2, c_1, r_1, \mathsf{cert}, \mathsf{sk})$ s.t. $((\mathsf{m}, c_2, \mathsf{PK}), w) \in \mathcal{R}$.

We construct $\mathcal{B}_{\mathsf{snark}}$, an adversary against simulation extractability ([28, Def. 2.10]) that behaves as follows:

Reduction 1: $\mathcal{B}(\mathsf{crs}_{\mathcal{R}}; r)$

- $\mathcal{B}_{\mathsf{snark}}$ activates $\mathcal{A}$ on pp and vk_{S} and proceeds as the challenger faithfully.
- When $\mathcal{A}$ outputs $(\mathsf{m}, c_2, \mathsf{PK})$ and proof, $\mathcal{B}_{\mathsf{snark}}$ forwards it to its challenger.

Note that for every PPT algorithm $\mathsf{snark}.\mathcal{X}_{\mathcal{B}}$ that can successfully produce a witness for a statement outputted by $\mathcal{B}_{\mathsf{snark}}$, it also produces a witness for the (same) statement outputted by $\mathcal{A}$. Assuming there is no such algorithm for the adversary $\mathcal{A}$, $\mathcal{B}_{\mathsf{snark}}$ would win simulation extractability with non-negligible probability.
$\mathsf{Hyb}_2^{\mathcal{A}} \to \mathsf{Hyb}_3^{\mathcal{A}}$. $\mathsf{Hyb}_3^{\mathcal{A}}$ is the same as Hyb_2, but the challenger additionally checks if $(\mathsf{cert}, \cdot) \in \mathcal{U}_h$.

The games are identically distributed conditioned on the checking $(\mathsf{cert}, \cdot) \in \mathcal{U}_h$ not failing. Assume it fails and consider $\mathcal{B}_{\mathsf{S}}$, an adversary against the unforgeability of S that acts as follows:

Reduction 2: $\underline{\mathcal{B}_{\mathsf{S}}(\mathsf{vk}_{\mathsf{S}})}$

- $\mathcal{A}$ acts as $\mathsf{Hyb}_2^{\mathcal{A}}(1^\lambda)$ challenger with $\mathsf{vk}_{\mathcal{KI}} := \mathsf{vk}_{\mathsf{S}}$. On $\mathsf{OHonJoin}$ queries, use the signing oracle.
- If any check fails output $\perp$.
- Let (m^*, π^*) be $\mathcal{A}$'s output and $(r_2, c_1, r_1, \mathsf{cert}, \mathsf{sk})$ the extracted witness. Parse $\mathsf{cert} = (\mathsf{vk}_i, \mathrm{ID}_i, \sigma)$ and output $(\mathsf{vk}_i \| \mathrm{ID}_i, \sigma)$.

Conditioned on $\mathcal{A}$ winning, the extractor outputs a valid witness. Moreover, since we assume that $(\mathsf{cert}, \cdot) \notin \mathcal{U}_h$, the final output of $\mathcal{B}_{\mathsf{S}}$ is a valid message-signature pair such that the signature was never queried to the signing oracle, meaning it succeeds in forging a signature.
Hyb_3 to One Wayness**:** Finally, let $\mathcal{A}$ be an adversary to $\mathsf{Hyb}_3^{\mathcal{A}}$, and let $q = \mathsf{poly}(\lambda)$ the number of $\mathsf{OHonJoin}$ queries.

Reduction 3: $\underline{\mathcal{B}_{\mathsf{OWF}}(1^\lambda, \mathsf{vk})}$

- Receive vk from her challenger and uniformly picks $c \leftarrow \{1, \ldots, q\}$.
- For all queries, $\mathcal{B}_{\mathsf{OWF}}$ answers as the challenger does, except that, in the i-th query of $\mathsf{OHonJoin}$, it returns the vk received.
- When $\mathcal{A}$ outputs (m^*, π^*), $\mathcal{B}$ parses $\pi^* = (c_2, \mathsf{proof})$ and runs $\mathsf{NIZK}.\mathcal{X}_{\mathcal{R}}(\mathsf{crs}_{\mathcal{R}})$.

– Let $(r_2, c_1, r_1, \mathsf{cert}, \mathsf{sk})$ be the extracted witness. If cert corresponds to the c-th $\mathsf{OHonJoin}$ query, $\mathcal{B}_{\mathsf{OWF}}$ outputs sk and $\perp$ otherwise.

Assuming that $\mathcal{A}$ wins in $\mathsf{Hyb}_3^{\mathcal{A}}$, cert corresponds to one of the q queries made to $\mathsf{OHonJoin}$. Since $\mathsf{vk} = \mathsf{vk}_c$ is sampled by the OWF challenger in the exact way as all the vk_i are sampled by $\mathcal{B}_{\mathsf{OWF}}$, $\mathcal{A}$ has no information about c. Thus, with probability $\frac{1}{q}$, cert corresponds to the vk and $f(\mathsf{sk}) = \mathsf{vk}$ and $\Pr\left[\mathsf{Hyb}_3^{\mathcal{A}}(1^\lambda) = 1\right] \leq q \cdot \Pr\left[\mathcal{B}_{\mathsf{OWF}}(1^{|x|}, f(x)) \in f^{-1}(f(x))\right]$. Since q is polynomial in λ, the probability of winning Hyb_3 is negligible.

We conclude that for all PPT $\mathcal{A}$, $\Pr\left[\mathsf{Exp\text{-}Unforge}_{\Pi\text{-}\mathsf{MAL},\mathcal{A}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$.

B.2 Proof of Thm 2

Proof. $\mathsf{Hyb}_0^{\mathcal{A}} \to \mathsf{Hyb}_1^{\mathcal{A}} \to \mathsf{Hyb}_2^{\mathcal{A}}$: In this case, $\mathsf{Hyb}_0^{\mathcal{A}}$ is the real Non-Frameability experiment (Fig. 3) instantiated with Π-MAL, and Hyb_1, Hyb_2 perform the same changes as in the case of unforgeability.
$\mathsf{Hyb}_2^{\mathcal{A}} \to \mathsf{Hyb}_3^{\mathcal{A}}$**:** $\mathsf{Hyb}_3^{\mathcal{A}}$ is the same as $\mathsf{Hyb}_2^{\mathcal{A}}$ but the challenger additionally checks if $c_1' = c_1$, and output 1 if this condition holds. Then, the games are identical if the decryption of the outer layer c_1 is equal to the element c_1' outputted by the extractor. Conditioned on the extractor succeeding, we can construct an adversary $\mathcal{B}_{\mathsf{TE}}$ against the Soundness of Threshold Encryption Scheme (Def. 10) as follows:

Reduction 4: $\underline{\mathcal{B}_{\mathsf{TE}}(\mathsf{pp}_{\mathsf{TE}})}$

– $\mathcal{B}_{\mathsf{TE}}$ initializes $\mathcal{A}$ as the challenger in $\mathsf{Hyb}_3^{\mathcal{A}}$ and uses the received parameters for the TE. It creates a public key for the TE by interacting with $\mathcal{A}$.
– Upon $\mathcal{A}$'s forgery $(\mathsf{m}, \pi, \mathsf{access}, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$, $\mathcal{B}_{\mathsf{TE}}$ parses $\mathsf{access} := (\mathsf{proof}_{\mathsf{m}}, \mathsf{m} \| c_1)$ and outputs $(\mathsf{m} \| c_1, \mathsf{proof}_{\mathsf{m}})$

Because we assume $\mathsf{snark}_{\mathcal{R}}.\mathcal{X}$ does not fail, we have $c_2 = \mathsf{TE.Enc}_{\mathsf{pk}_{\mathsf{TE}}^{\mathcal{GU}}}(\mathsf{m} \| c_1'; r_2)$. If $\mathsf{m} \| c_1 \neq \mathsf{m} \| c_1'$, $\mathcal{B}_{\mathsf{TE}}$ breaks soundness of TE (Def. 10), and thus we conclude that for all $\mathcal{A}$, there exists a negligible function μ such that $\left|\Pr\left[\mathsf{Hyb}_2^{\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hyb}_3^{\mathcal{A}}(1^\lambda) = 1\right]\right| \leq \mu(\lambda)$
$\mathsf{Hyb}_3^{\mathcal{A}} \to \mathsf{Hyb}_4^{\mathcal{A}}$. $\mathsf{Hyb}_4^{\mathcal{A}}$**.** This is $\mathsf{Hyb}_3^{\mathcal{A}}$ with the additional check $\mathsf{cert}' = \mathsf{cert}_{\mathrm{ID}}$; the final output is 1 if this equality also holds. Note that $\mathsf{Hyb}_3^{\mathcal{A}}$ and $\mathsf{Hyb}_4^{\mathcal{A}}$ are identical conditioned on $\mathsf{cert} = \mathsf{cert}'$. If this is not the case, we can construct an adversary $\mathcal{B}_{\mathsf{NIZK}}$ against simulation extractability that acts as follows:

Reduction 5: $\underline{\mathcal{B}_{\mathsf{NIZK}}(\mathsf{crs}_{\mathcal{VD}})}$

– $\mathcal{B}_{\mathsf{NIZK}}$ initializes $\mathcal{A}$ as the challenger in $\mathsf{Hyb}_4^{\mathcal{A}}$.
– Upon $\mathcal{A}$'s output $(\mathsf{m}, \pi, \mathsf{access}, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$, $\mathcal{B}_{\mathsf{NIZK}}$ outputs $((\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, c_1, \mathsf{cert}_{\mathrm{ID}}), \mathsf{proof}_{\mathrm{ID}})$.

Because $(c_1', \mathsf{cert}', r_1)$ is part of a valid witness for $\mathcal{R}$, $c_1 = c_1' = \mathsf{E.Enc}_{\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}}(\mathsf{cert}'; r_1)$. If $\mathsf{cert}' \neq \mathsf{cert}_{\mathrm{ID}}$, then $(\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}, c_1, \mathsf{cert}_{\mathrm{ID}}) \notin \mathcal{VD}$ and there is no witness for this statement; thus, every extractor fails. We conclude the games are indistinguishable.
Hyb_4 to One Wayness**:** Finally recall that, from the previous game, we have $\mathsf{cert}' = \mathsf{cert}_{\mathrm{ID}}$ and, from the real game, the winning condition of $\mathcal{A}$ implies $(\mathsf{cert}_{\mathrm{ID}}, \cdot) \in \mathcal{U}_h$. Conditioned on this two facts, the reduction from Hyb_4 to One Wayness is the same as the reduction from Hyb_3 to One Wayness in Theorem 1.

We conclude that, for all PPT $\mathcal{A}$, $\Pr\left[\mathsf{Exp\text{-}UserNonFrame}_{\Pi\text{-}\mathsf{MAL},\mathcal{A}}(1^\lambda) = 1\right] \leq \mu(\lambda)$.

B.3 Proof of Thm 3

Proof. $\mathsf{Hyb}_0^{\mathcal{A}} \to \mathsf{Hyb}_1^{\mathcal{A}} \to \mathsf{Hyb}_2^{\mathcal{A}} \to \mathsf{Hyb}_3^{\mathcal{A}}$. $\mathsf{Hyb}_0^{\mathcal{A}}$**.** is the real accountability game of Def. 4 instantiated with Π-MAL; $\mathsf{Hyb}_1, \mathsf{Hyb}_2, \mathsf{Hyb}_3$ apply the same sequence of changes as in the proof of Thm. 2.

$\mathsf{Hyb}_3^{\mathcal{A}} \to \mathsf{Hyb}_4^{\mathcal{A}}$. The latter is the same as $\mathsf{Hyb}_3^{\mathcal{A}}$ except it also checks: $\mathsf{cert}' = \mathsf{cert}$ and the output is 1 if and only if this condition also holds. Note that here we simply rely on correctness of the encryption scheme: cert is the decryption of c_1 by construction and cert' is a part of a valid witness for $\mathcal{R}$ so, with non-negligible probability, it is a decryption of c_1 and then $\mathsf{cert} = \mathsf{cert}'$.
Hyb_4 to Unforgeability**:** Next, we show that an adversary $\mathcal{A}$ winning $\mathsf{Hyb}_4^{\mathcal{A}}$, implies an adversary $\mathcal{B}_{\mathsf{S}}$ that breaks the unforgeability of the signature scheme $\mathcal{B}_{\mathsf{S}}$ works as follows:

Reduction 6: $\underline{\mathcal{B}_{\mathsf{S}}(\mathsf{vk}_{\mathsf{S}})}$

- Initialize $\mathcal{A}$ by acting as $\mathsf{Hyb}_4^{\mathcal{A}}(1^\lambda)$; set $\mathsf{vk}_{\mathcal{KI}} := \mathsf{vk}_{\mathsf{S}}$ and use the signing oracle to simulate $\mathsf{OHonJoin}$.
- $\mathcal{A}$ outputs a pair (m^*, π^*).
- Let $(r_2, c_1, r_1, \mathsf{cert}', \mathsf{sk})$ be the extracted witness, parse $\mathsf{cert}' = (\mathsf{vk}_i, \mathrm{ID}_i, \sigma)$ and output $(\mathsf{vk}_i \| \mathrm{ID}_i, \sigma)$.

Assuming $\mathcal{A}$ wins $\mathsf{Hyb}_4^{\mathcal{A}}$, the pair $(\mathsf{vk}_i \| \mathrm{ID}_i, \sigma)$ is a valid message-signature pair under the key vk_{S}, as we already have the extractor outputs a valid witness. Furthermore, we are guaranteed that the honestly computed cert, does not belong to the set $\mathcal{U}_h \cup \mathcal{U}_m$ which implies that $\mathsf{cert}' = \mathsf{cert} \notin \mathcal{U}_h \cup \mathcal{U}_m$. Thus, $(\mathsf{vk}_i \| \mathrm{ID}_i, \sigma)$ was not a query-answer pair to the signing oracle, so it constitutes a valid forgery. We conclude that, for all PPT $\mathcal{A}$, $\Pr\left[\mathsf{t\text{-}Exp\text{-}UserAccountability}_{\Pi\text{-}\mathsf{MAL},\mathcal{A}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$.

B.4 Proof of Thm. 4

Proof. Assume an adversary $\mathcal{A}$ against Manager Non-Frameability (Def. 6). We construct an adversary $\mathcal{B}_{\mathsf{S}}$ against the Signature Scheme.

Reduction 7: $\underline{\mathcal{B}_{\mathsf{S}}(\mathsf{pp}_{\mathsf{S}}, \mathsf{vk}_{\mathsf{S}})}$

- $\mathcal{B}_{\mathsf{S}}$ answers all oracles honestly, using its secret keys, except for OReq which it answers as follows: On input (m_i, π_i), $\mathcal{B}_{\mathsf{S}}$ queries it's signing oracle with $\mathsf{m}_i \| \pi_i$ and gets a signature σ_i. It gives $\mathsf{req}_i := \sigma_i$ to $\mathcal{A}$ and updates the request set $\mathcal{Q}$ with $(\mathsf{m}_i, \pi_i, \mathsf{req}_i)$.
- Upon receiving $(\mathsf{pk}^{\mathcal{GU}}, \mathsf{m}, \pi, \mathsf{req}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{pk}^{\mathcal{KI}}, \mathsf{pk}^{\mathcal{GM}})$ from $\mathcal{A}$, outputs $(\mathsf{m} \| \pi, \mathsf{req})$ as its forgery.

Assume $\mathcal{A}$ wins Manager Non-Frameability. Then, $(\mathsf{m}, \pi, \mathsf{req})$ satisfies: (i) $\mathsf{JudgeReq}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{req}) = 1$ and (ii) $(\mathsf{m}, \pi, \mathsf{req}) \notin \mathcal{Q}$. Noting that $\mathsf{JudgeReq}$ is simply signature verification, and that $\mathcal{Q}$ contains all the queries $\mathcal{B}_{\mathsf{S}}$ made to its Signing Oracle, we conclude that $\mathcal{B}_{\mathsf{S}}$ succeeds in forging a signature, thus, for all PPT $\mathcal{A}$, $\Pr\left[\mathsf{Exp\text{-}ManagerNonFrame}_{\Pi\text{-}\mathsf{MAL},\mathcal{A}}(1^\lambda) = 1\right] \leq \mu(\lambda)$.

B.5 Proof of Thm. 5

Proof. **Case 1: Malicious group manager.** For the first case, we apply the following sequence of hybrids. $\mathsf{Hyb}_0^{\mathcal{A}} \to \mathsf{Hyb}_1^{\mathcal{A}}$: $\mathsf{Hyb}_0^{\mathcal{A}}$ is the real $\mathsf{Exp\text{-}UAnon\&MalGM}$ game from Fig. 6, instantiated with Π-MAL. $\mathsf{Hyb}_1^{\mathcal{A}}$ is the same except that when $\mathcal{A}$ sends its challenge $(\mathsf{m}, \mathsf{cert}_0, \mathsf{cert}_1)$, we compute the response $\pi := (c_2, \mathsf{proof})$ by replacing proof with $\mathsf{proof}_{\mathsf{Sim}} \leftarrow \mathsf{snark}_{\mathcal{R}}.\mathsf{Sim}$. By the Zero Knowledge property of the snark the proofs are identically distributed and, thus, so are the two hybrid games.
$\mathsf{Hyb}_1^{\mathcal{A}} \to \mathsf{Hyb}_2^{\mathcal{A}}$**.** $\mathsf{Hyb}_2^{\mathcal{A}}$ is the same as $\mathsf{Hyb}_1^{\mathcal{A}}$ with the following change: When replying to $\mathsf{OGrant}(\mathsf{m}, \pi, \mathsf{req})$, parse $\pi = (c_2, \mathsf{proof})$; if $(\mathsf{m}, \pi, \cdot) \notin \mathcal{T}$, run $\mathsf{snark}_{\mathcal{R}}.\mathcal{X}(\mathsf{crs})$. Let $(r_2', c_1', r_1, \mathsf{cert}, \mathsf{sk})$ be the output and abort if $((\mathsf{m}, c_2, \mathsf{PK}), (r_2', c_1', r_1, \mathsf{cert}, \mathsf{sk})) \notin \mathcal{R}$.

We use a standard hybrid argument for proving both games are indistinguishable. Let Q be an upper bound on the queries made by $\mathcal{A}$ on OGrant. For $i \in \{0, \ldots, Q\}$, let $\mathsf{Hyb}_{1,i}^{\mathcal{A}}$ be the same as $\mathsf{Hyb}_1^{\mathcal{A}}$ with the following change: for the first i queries to OGrant, (i) let m, π be the query. Parse $\pi = (c_2, \mathsf{proof})$. If $(\mathsf{m}, \pi, \cdot) \notin \mathcal{T}$, run $\mathsf{snark}_{\mathcal{R}}.\mathcal{X}(\mathsf{crs})$. Let $(r_2', c_1', r_1, \mathsf{cert}, \mathsf{sk})$ be the output; (ii) abort if for any witness outputted by $\mathsf{snark}_{\mathcal{R}}.\mathcal{X}$, $\mathcal{R}((\mathsf{m}, c_2, \mathsf{PK}), (r_2', c_1', r_1, \mathsf{cert}, \mathsf{sk})) = 0$. By construction, $\Pr\left[\mathsf{Hyb}_1^{\mathcal{A}}(1^\lambda) = 1\right] = \Pr\left[\mathsf{Hyb}_{1,0}^{\mathcal{A}}(1^\lambda) = 1\right]$ and $\Pr\left[\mathsf{Hyb}_2^{\mathcal{A}}(1^\lambda) = 1\right] = \Pr\left[\mathsf{Hyb}_{1,Q}^{\mathcal{A}}(1^\lambda) = 1\right]$.

$\mathsf{Hyb}_{1,i-1}, \mathsf{Hyb}_{1,i-1}$ can differ only if in the i-th OGrant query, we attempt to extract a satisfying witness. Conditioned on this step being successful, the games are identical. If this is not the case, we can break simulation extractability as in the case of Th. 1

Thus, by simulation extractability of $\mathsf{snark}_{\mathcal{R}}$, $\mathsf{Hyb}^{\mathcal{A}}_{1,i-1}$ and $\mathsf{Hyb}^{\mathcal{A}}_{1,i}$ are indistinguishable for all $i \in \{1, \ldots, Q\}$. In particular, it is the case for $\mathsf{Hyb}^{\mathcal{A}}_{1,0}$ ($\mathsf{Hyb}^{\mathcal{A}}_{1}$) and $\mathsf{Hyb}^{\mathcal{A}}_{1,Q}$ ($\mathsf{Hyb}^{\mathcal{A}}_{2}$).

$\mathsf{Hyb}^{\mathcal{A}}_{2} \rightarrow \mathsf{Hyb}^{\mathcal{A}}_{3}$**.** $\mathsf{Hyb}^{\mathcal{A}}_{3}$ is the same as $\mathsf{Hyb}^{\mathcal{A}}_{2}$ with the following changes: (i) when $\mathcal{A}$ queries $\mathsf{OMemberAuth}$ on $(\mathsf{m}, \mathsf{cert})$, we keep with the query kept in $\mathcal{T}$ the randomness r_2 used to compute $\mathsf{TE.Enc}_{\mathsf{pk}^{\mathcal{GU}}}(\mathsf{m} \| c_1; r_2)$. (ii) to answer to OGrant, use the decryption simulator $\mathcal{S}$ to output $(\mathsf{proof}_{\mathsf{m}}, \mathsf{m} || c_1) \leftarrow \mathsf{TE.ProveValidDec}(\mathcal{A}(\mathsf{st}_{\mathcal{A}}), \mathcal{S}(\mathsf{m} \| c_1, r_2))(c_2)$, where $(\mathsf{m} \| c_1, r_2)$ is either in the set $\mathcal{T}$, or by the invocation of $\mathsf{snark}_{\mathcal{R}}.\mathcal{X}_{\mathcal{A}}$.

Let Q be an upper bound on the queries made by $\mathcal{A}$ on OGrant. For $i \in \{0, \ldots, Q\}$, let $\mathsf{Hyb}^{\mathcal{A}}_{2,i}$ be the same as $\mathsf{Hyb}^{\mathcal{A}}_{2}$ with the following change: for the first i queries to OGrant, (i) let m, π be the query. Parse $\pi = (c_2, \mathsf{proof})$ and $\mathsf{m} \| c_1, r_2$ be the corresponding decryption obtained by either computing π, or by querying the extractor; (ii) use the decryption simulator $\mathcal{S}$ and output $(\mathsf{proof}_{\mathsf{m}}, \mathsf{m} || c_1') \leftarrow \mathsf{TE.ProveValidDec}(\mathcal{A}(\mathsf{st}_{\mathcal{A}}), \mathcal{S}(\mathsf{m} \| c_1, r_2))(c_2)$ by using $(\mathsf{m} \| c_1, r_2)$ that is either in the set $\mathcal{T}$, or from the invocation of $\mathsf{snark}_{\mathcal{R}}.\mathcal{X}_{\mathcal{A}}$. Observe that $\Pr\left[\mathsf{Hyb}^{\mathcal{A}}_{2}(1^\lambda) = 1\right] = \Pr\left[\mathsf{Hyb}^{\mathcal{A}}_{2,0}(1^\lambda) = 1\right]$ and $\Pr\left[\mathsf{Hyb}^{\mathcal{A}}_{3}(1^\lambda) = 1\right] = \Pr\left[\mathsf{Hyb}^{\mathcal{A}}_{2,Q}(1^\lambda) = 1\right]$.

Also, note that the only difference between $\mathsf{Hyb}^{\mathcal{A}}_{2,i-1}$ and $\mathsf{Hyb}^{\mathcal{A}}_{2,i}$ is the way the proof proof is computes: in $\mathsf{Hyb}^{\mathcal{A}}_{2,i-1}$, it is computed by executing $\mathsf{TE.ProveValidDec}$ with the honest users, who have as inputs the corresponding secret keys, while in $\mathsf{Hyb}^{\mathcal{A}}_{2,i}$ it is computed by executing $\mathsf{TE.ProveValidDec}$ with the simulator $\mathcal{S}$ who only knows the decryption and the randomness used. Since these distributions are identical by Decryption Simulatability, the games are indistinguishable. Because Q is polynomial in λ, this in particular means that $\mathsf{Hyb}^{\mathcal{A}}_{2}$ ($\mathsf{Hyb}^{\mathcal{A}}_{2,0}$) and $\mathsf{Hyb}^{\mathcal{A}}_{3}$ ($\mathsf{Hyb}^{\mathcal{A}}_{2,Q}$) are indistinguishable.

Hyb_3 *to* (t, n)*-Indistinguishability.* Finally, assuming $\mathcal{A}$ wins Hyb_3, we create an adversary $\mathcal{B}_{\mathsf{TE}}$ that behaves as follows:

Reduction 8: $\underline{\mathcal{B}_{\mathsf{TE}}}$

- $\mathcal{B}_{\mathsf{TE}}$ receives the parameters for the TE and activates $\mathcal{A}$, participating with $\mathcal{A}$ to create keys for the TE. For any requests to OGrant, $\mathcal{B}_{\mathsf{TE}}$ acts as the challenger in $\mathsf{Hyb}^{\mathcal{A}}_{3}$.
- On $\mathcal{A}$'s challenge $(\mathsf{pk}^{\mathcal{GM}}, \mathsf{m}, \mathsf{cert}_0, \mathsf{cert}_1)$, $\mathcal{B}_{\mathsf{TE}}$ sets $\mathsf{m}_0 = \mathsf{m} \| \mathsf{Enc}_{\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}}(\mathsf{cert}_0)$ and $\mathsf{m}_1 = \mathsf{m} \| \mathsf{Enc}_{\mathsf{pk}_{\mathsf{E}}^{\mathcal{GM}}}(\mathsf{cert}_1)$ as its own challenge.
- Gets $c = \mathsf{TE.Enc}_{\mathsf{pk}_{\mathsf{TE}}}(\mathsf{m}_b)$ and gives $\pi := (c, \mathsf{proof})$ to $\mathcal{A}$, with proof computed as in Hyb_3.
- $\mathcal{B}_{\mathsf{TE}}$ outputs the same value $\mathcal{A}$ outputs.

We have that $\mathcal{B}_{\mathsf{TE}}$ wins the indistinguishability game iff $\mathcal{A}$ wins in $\mathsf{Hyb}^{\mathcal{A}}_{3}$. So we conclude that, for all PPT adversaries $\mathcal{A}$, $\left|\Pr\left[\mathsf{Hyb}^{\mathcal{A}}_{0}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Hyb}^{\mathcal{A}}_{3}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$, $\Pr\left[\mathsf{Hyb}^{\mathcal{A}}_{3}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$.

Case 2: Malicious guardians. In this case, the adversary fully controls the guardians. We proceed analogously to the previous case by applying the following sequence of hybrids.

$\mathsf{Hyb}^{\mathcal{A}}_{0} \rightarrow \mathsf{Hyb}^{\mathcal{A}}_{1} \rightarrow \mathsf{Hyb}^{\mathcal{A}}_{2}$**.** $\mathsf{Hyb}^{\mathcal{A}}_{0}$ is the real $\mathsf{t\text{-}Exp\text{-}UAnon\&MalGuard}$ game from Fig. 7, instantiated with Π-MAL; $\mathsf{Hyb}^{\mathcal{A}}_{1}$ and $\mathsf{Hyb}^{\mathcal{A}}_{2}$ produce the same changes as in anonymity against a Malicious group manager so we refer the reader to the former case for the reductions.

$\mathsf{Hyb}^{\mathcal{A}}_{2} \rightarrow \mathsf{Hyb}^{\mathcal{A}}_{3}$**.** $\mathsf{Hyb}^{\mathcal{A}}_{3}$ is the same as $\mathsf{Hyb}^{\mathcal{A}}_{2}$ but to answer to $\mathsf{ODeanon}$ on input $(\mathsf{m}, \pi, \mathsf{access})$, instead of running $\mathsf{cert} \leftarrow \mathsf{E.Dec}_{\mathsf{sk}_{\mathsf{E}}}(c_1)$, (1) use the value cert that is either saved in $\mathcal{T}$ if $(\mathsf{m}, \pi, \mathsf{cert}) \in \mathcal{T}$, or computed by $\mathsf{snark}_{\mathcal{R}}.\mathcal{X}$, (2) run $\mathsf{proof} \leftarrow \mathsf{NIZK}_{\mathcal{VD}}.\mathsf{Sim}(\tau_{\mathcal{VD}}, (\mathsf{pk}_{\mathsf{E}}, \mathsf{cert}, c_1))$, (3) output $(\mathsf{cert}, \mathsf{proof})$. We can argue exactly as in the corresponding games with the relation $\mathcal{R}$ of the malicious manager case that the games are indistinguishable.

Hyb_3 *to Indistinguishability.* Finally, we can create a reduction from Hyb_3 to Indistinguishability of Encryption.

Reduction 9: $\underline{\mathcal{B}_{\mathsf{E}}}$

- $\mathcal{B}_{\mathsf{E}}$ receives the parameters pp and a public key pk_{E} for E. It creates the other parameters for the protocol.
- $\mathcal{B}_{\mathsf{E}}$ activates $\mathcal{A}$ and acts as the challenger for $\mathsf{Hyb}_3^{\mathcal{A}}$.
- When $\mathcal{A}$ gives its challenge $(\mathsf{pk}^{\mathcal{GU}}, \mathsf{m}, \mathsf{cert}_0, \mathsf{cert}_1)$, then $\mathcal{B}_{\mathsf{E}}$ sets $\mathsf{m}_0 = \mathsf{cert}_0$ and $\mathsf{m}_1 = \mathsf{cert}_1$, as its own challenge.
- It gets $c = \mathsf{E.Enc}_{\mathsf{pk}_{\mathsf{E}}}(\mathsf{m}_b)$ and computes $c_2 \leftarrow \mathsf{TE.Enc}_{\mathsf{pk}^{\mathcal{GU}}}(\mathsf{m} \| c)$. It computes π as in Hyb_3 (by using the NIZK simulator). It sets $\pi := (c_2, \mathsf{proof})$ and gives π to $\mathcal{A}$.
- $\mathcal{B}_{\mathsf{TE}}$ outputs the same value $\mathcal{A}$ outputs.

We have that $\mathcal{B}_{\mathsf{E}}$ wins the indistinguishability game iff $\mathcal{A}$ wins in $\mathsf{Hyb}_3^{\mathcal{A}}$. We conclude that, for all PPT adversaries $\mathcal{A}$,

$$\Pr\left[\mathsf{Exp\text{-}UAnon\&MalGM}_{\Pi\text{-}\mathsf{MAL},\mathcal{A}_1}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

$$\Pr\left[\mathsf{t\text{-}Exp\text{-}UAnon\&MalGuard}_{\Pi\text{-}\mathsf{MAL},\mathcal{A}_2}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

C Security of the DGS based construction

C.1 Proof of Thm. 6

Proof. To prove unforgeability we proceed by hybrids to modify the $\mathsf{Exp\text{-}Unforge}$ game.

- Hyb_0 is the $\mathsf{Exp\text{-}Unforge}$ game.
- Hyb_1 is the same as Hyb_0 with the following differences: (1) we use simulated proofs for $\mathsf{NIZK}_{\mathsf{ddh}}$ and $\mathsf{NIZK}_{\mathsf{Enc}}$ and (2) when the adversary queries $\mathsf{OMemberAuth}(\mathsf{m}, \mathsf{cert}_{\mathrm{ID}})$ we simulate the signature of knowledge.
- Hyb_2 is the same as Hyb_1 except that, when the adversary outputs the forgery $(\mathsf{m}, \pi = (\sigma_1', \sigma_2', \pi_{\mathsf{PS}}))$, if (m, π) was not queried to $\mathsf{OMemberAuth}$, we also run the extractor of $\mathsf{NIZK}_{\mathsf{PS}}$ to extract a witness for σ_1', σ_2', and output $\perp$ if the extraction fails.
- Hyb_3 is the same as Hyb_2 except that if the extracted witness contains a message/signature pair $\mathsf{sk}, \pi = (\sigma_1, \sigma_2, \pi_{\mathsf{PS}})$ for which $(\cdot, \mathsf{sk}) \notin \mathcal{U}_h$ the game outputs $\perp$ and aborts.

Hyb_0 and Hyb_1 have outputs that are statistically close by the zero knowledge property of the underling NIZKs. Similarly, Hyb_1 and Hyb_2 proceed identically unless the extraction fails. This happens only with negligible probability by simulation extractability of $\mathsf{NIZK}_{\mathsf{PS}}$. Finally, $\mathsf{Hyb}_2, \mathsf{Hyb}_3$ proceed identically unless the extractor outputs some message/signature pair which does not correspond to an honest user's secret.

Assume for the sake of contradiction that the extractor outputs a message/signature pair that does not correspond to an honest user's secret with non-negligible probability.

Then we show how to construct a a reduction $\mathcal{B}$ that can break unforgeability of PS signature scheme.

$\mathcal{B}$ takes input parameters gk and a PS signature public key (X, Y_0, Y_1), it uses $\mathcal{A}$ and proceeds as in Hyb_3 except that when $\mathcal{A}$ makes a query for adding a user with ID, it samples a secret sk and queries its signing oracle to get the signature (a, σ_1, σ_2) and programs $\mathcal{H}$ to output $(a, \sigma_1) \leftarrow \mathcal{H}(\mathrm{ID})$. Finally, when the adversary $\mathcal{A}$ outputs the forgery (m, π), $\mathcal{B}$ runs the extractor and gets a message/signature pair $\mathsf{sk}, (a, \sigma_1, \sigma_2)$ which it outputs as its forgery. By assumption, the signing oracle never answered a query on sk, and by simulation extractability of $\mathsf{NIZK}_{\mathsf{PS}}$ the extractor outputs a valid witness, namely, a verifying message/signature pair w.r.t. (X, Y_0, Y_1), so $\mathcal{B}$ succeeds in making the forgery. Since the PS signature scheme is secure under q-MSDH assumption, where q is the number of signing queries made and q corresponds to the number of honest users requested by $\mathcal{A}$, the PS forgery happens with negligible probability under n_U-MSDH. .

Finally, if the SDL assumption holds, no adversary $\mathcal{A}$ can win Hyb_3 with non-negligible probability. Assume otherwise and let $\mathcal{A}$ be such an adversary. We build an adversary $\mathcal{B}$ solving SDL with non-negligible probability. $\mathcal{B}$ works as follows:

– $\mathcal{B}$ takes as input $\mathsf{gk}, (g_1, h_1, g_2, h_2) = (g, h, g^\mu, h^\mu)$ and uses gk, g, h as the public parameters for the scheme. Let x, y_0, y_1 be the PS signature signing key and X, Y_0, Y_1 the corresponding verification key sampled in Hyb_3.
– It samples $i \in [n_U]$.
– It answers oracle queries of $\mathcal{A}$ as follows:
  • $\mathsf{OHonJoin}$. It onlyanswers honestly except for the query ID_i. In that case, if such a query has not been explicitly made, it first queries $\mathcal{H}(\mathrm{ID}_i) = (a, g^r)$ and then samples $\mathsf{sk}_1, \mathsf{sk}_2 \leftarrow \mathbb{Z}_p^*$. It then implicitly sets as the secret key of this user the value $\mu(\mathsf{sk}_1 + \mathsf{sk}_2)$ as follows: it sets $\hat{g} = g^r$ and
    * $g_{\mathsf{sk}_1} = g_2^{\mathsf{sk}_1} = g^{\mu \mathsf{sk}_1}$, $g_{\mathsf{sk}_2} = g_2^{\mathsf{sk}_2} = g^{\mu \mathsf{sk}_2}$.
    * $\hat{g}_{\mathsf{sk}_1} = g_2^{r\mathsf{sk}_1} = g^{r\mu \mathsf{sk}_1} = \hat{g}^{\mu \mathsf{sk}_1}$, $\hat{g}_{\mathsf{sk}_2} = g_2^{r\mathsf{sk}_2} = g^{r\mu \mathsf{sk}_2} = \hat{g}^{\mu \mathsf{sk}_2}$.

    It then computes the secret shares for the guardians and group manager using the values h_1, h_2. It first samples uniform coefficients $p_1, \ldots, p_t \in \mathbb{Z}_p$, computes the commitments $P_i = \hat{g}^{p_i} = g_2^{rp_i}$, computes the shares

$$Y_0^{p(\ell)} := h_2^{y_0 \mathsf{sk}_1} h^{y_0 \sum p_i \ell^i} =$$
$$h^{\mu y_0 \mathsf{sk}_1} h^{y_0 \sum p_i \ell^i} = h^{y_0(\mu \mathsf{sk}_1 + \sum p_i \ell^i)} = h^{y_0 p(\ell)} \text{ for } \mathsf{C}_\ell \in \mathcal{GU}$$

$$Y_0^{\mu \mathsf{sk}_2} := h_2^{y_0 \mathsf{sk}_2} = h^{\mu y_0 \mathsf{sk}_2}$$

    It keeps in its state the values $\sigma_1 = \hat{g} = g^r$, $\sigma_2 = g^{r(x + y_0 \mu(\mathsf{sk}_1 + \mathsf{sk}_2) + y_1 a)}$ to later answer $\mathsf{OMemberAuth}$ queries. The latter can be computed given $g^\mu = g_2$.
  • $\mathsf{OMemberAuth}$. It works honestly as in Hyb_3. Specifically, for the signatures of knowledge issued for user i, it samples $\alpha \leftarrow \mathbb{Z}_p$, computes $(\sigma_1', \sigma_2') = (\sigma_1^\alpha, \sigma_2^\alpha)$ and simulates a PoK for $\mathsf{NIZK}_{\mathsf{PS}}$, that is a PoK of $(\mu(\mathsf{sk}_1 + \mathsf{sk}_2), a)$ such that $e(\sigma_1', Y_0)^m e(\sigma_1', Y_1)^a = e(\sigma_2', h) e(\sigma_1', X)^{-1}$.

When $\mathcal{A}$ outputs a forgery m, π, the extractor outputs a witness which corresponds to one of the secret keys of an honest user by assumption. With probability at least $\frac{1}{n_U}$ this corresponds to user i. If this is the case, $\mathcal{B}$ gets a message signature pair $(\mathsf{sk}, \cdot)$ where $\mathsf{sk} = \mu(\mathsf{sk}_1 + \mathsf{sk}_2)$ from which it can compute μ such that $g_2 = g_1^\mu$ and $h_2 = h_1^\mu$, thus solving the SDL challenge.

C.2 Proof of Thm. 7

Proof. We proceed a sequence of hybrids that modifies the $\mathsf{Exp\text{-}UserNonFrame}$ game.

– Hyb_0 is the $\mathsf{Exp\text{-}UserNonFrame}$ game.
– Hyb_1 is the same as Hyb_0 except the game: (1) uses simulated proofs for $\mathsf{NIZK}_{\mathsf{ddh}}$ and $\mathsf{NIZK}_{\mathsf{Enc}}$ and (2) simulates the proof of knowledge $\mathsf{NIZK}_{\mathsf{PS}}$ when the adversary queries $\mathsf{OMemberAuth}(\mathsf{m}, \mathsf{cert}_{\mathrm{ID}})$.
– Hyb_2 is the same as Hyb_1 except that when the adversary outputs $(\mathsf{m}, \pi, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$, we also run the extractor of $\mathsf{NIZK}_{\mathsf{PS}}$ to extract a witness for σ_1', σ_2', and output $\perp$ if the extraction fails.
– Hyb_3 is the same as Hyb_2 except that we output $\perp$ and abort if we run the extractor and extract a witness that contains a value $(\cdot, \mathsf{sk}) \in \mathcal{U}_h$.
– Hyb_4 is the same as Hyb_3 except that when the adversary $\mathcal{A}$ outputs $(\mathsf{m}, \pi, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$ where $\mathsf{proof}_{\mathrm{ID}} = \left(\{B_{i,j_k}, \pi_{i,j_k}\}_{k=1}^{t+1}, B_i, \pi_i\right)$ we also run the extractor of $\mathsf{NIZK}_{\mathsf{Dec}}$ for the statement corresponding to π_{i,j_k}, π_i to extract witnesses z_{j_k}, z. Hyb_4 outputs $\perp$ if any extraction fails to extract a valid witness.

Hyb_0 and Hyb_1 have outputs that are statistically close by the knowledge soundness property of $\mathsf{NIZK}_{\mathsf{Dec}}$. Hyb_1 and Hyb_2 proceed identically by the zero-knowledge property of the underlying NIZKs. We also claim that Hyb_2 and Hyb_3 proceed identically, or equivalently Hyb_3 does not abort.

To show Hyb_3 does not abort, we show that we never extract a witness corresponding to an honest user if this has not been queried to $\mathsf{OMemberAuth}$. The proof is the same as the last case of the unforgeability property.

Assume that there exists an adversary that, with non-negligible probability causes Hyb_3 to abort conditioned on the extractor not failing. We construct an adversary $\mathcal{B}$ against the SDL assumption. $\mathcal{B}$ works as follows:

– $\mathcal{B}$ takes as input $\mathsf{gk}, (g_1, h_1, g_2, h_2) = (g, h, g^\mu, h^\mu)$ and uses gk, g, h as the public parameters for the scheme. Let x, y_0, y_1 be the PS signature signing key and X, Y_0, Y_1 the corresponding verification key sampled in Hyb_3.
– It samples $i \in [n_U]$.
– It answers oracle queries of $\mathcal{A}$ as follows:
  • $\mathsf{OHonJoin}$ only answers honestly except for the query ID_i. For this query, it first queries $\mathcal{H}(\mathrm{ID}_i) = (a, g^r)$ if such a query has not been made yet, then it samples $\mathsf{sk}_1, \mathsf{sk}_2 \leftarrow \mathbb{Z}_p^*$. It then implicitly sets as the secret key of this user the value $\mu(\mathsf{sk}_1 + \mathsf{sk}_2)$ as follows: it sets $\hat{g} = g^r$ and
    1. $g_{\mathsf{sk}_1} = g_2^{\mathsf{sk}_1} = g^{\mu \mathsf{sk}_1}$, $g_{\mathsf{sk}_2} = g_2^{\mathsf{sk}_2} = g^{\mu \mathsf{sk}_2}$.
    2. $\hat{g}_{\mathsf{sk}_1} = g_2^{r\mathsf{sk}_1} = g^{r\mu \mathsf{sk}_1} = \hat{g}^{\mu \mathsf{sk}_1}$, $\hat{g}_{\mathsf{sk}_2} = g_2^{r\mathsf{sk}_2} = g^{r\mu \mathsf{sk}_2} = \hat{g}^{\mu \mathsf{sk}_2}$.

    It then computes the secret shares for the guardians, group manager using the values h_1, h_2. It first samples uniform coefficients $p_1, \ldots, p_t \in \mathbb{Z}_p$, computes the commitments $P_i = \hat{g}^{p_i} = g_2^{r p_i}$, computes the shares

$$Y_0^{p(\ell)} := h_2^{y_0 \mathsf{sk}_1} h^{y_0 \sum p_i \ell^i} = h^{\mu y_0 \mathsf{sk}_1} h^{y_0 \sum p_i \ell^i} = h^{y_0 (\mu \mathsf{sk}_1 + \sum p_i \ell^i)} = h^{y_0 p(\ell)} \text{ for } \mathsf{C}_\ell \in \mathcal{GU}$$

$$Y_0^{\mu \mathsf{sk}_2} := h_2^{y_0 \mathsf{sk}_2} = h^{\mu y_0 \mathsf{sk}_2}$$

    It keeps in its state the values $\sigma_1 = \hat{g} = g^r$, $\sigma_2 = g^{r(x + y_0 \mu(\mathsf{sk}_1 + \mathsf{sk}_2) + y_1 a)}$ to later answer $\mathsf{OMemberAuth}$ queries. The latter can be computed given $g^\mu = g_2$.
  • $\mathsf{OMalJoin}$. It works honestly as in Hyb_3.
  • $\mathsf{OMemberAuth}$. It works honestly as in Hyb_3. Specifically, for the signatures of knowledge issued for user i, it samples $\alpha \leftarrow \mathbb{Z}_p$, computes $(\sigma_1', \sigma_2') = (\sigma_1^\alpha, \sigma_2^\alpha)$ and simulates a PoK for $\mathsf{NIZK}_{\mathsf{PS}}$, that is a PoK of $\mu(\mathsf{sk}_1 + \mathsf{sk}_2), a$ such that $e(\sigma_1', Y_0)^m e(\sigma_1', Y_1)^a = e(\sigma_2', h) e(\sigma_1', X)^{-1}$.

When $\mathcal{A}$ outputs a forgery (m, π), the extractor outputs a witness which corresponds to one of the secret keys of an honest user by assumption. With probability at least $\frac{1}{n_U}$ this corresponds to user i. If this is the case, $\mathcal{B}$ gets a message signature pair $(\mathsf{sk}, \cdot)$ where $\mathsf{sk} = \mu(\mathsf{sk}_1 + \mathsf{sk}_2)$ from which it can compute μ such that $g_2 = g_1^\mu$ and $h_2 = h_1^\mu$, thus solving the SDL challenge.

Hyb_3 and Hyb_4 are indistinguishable due to knowledge soundness of $\mathsf{NIZK}_{\mathsf{Dec}}$.

Finally, no adversary can win Hyb_4 except with negligible probability. Let $\mathsf{cert}_i := \big(\mathrm{ID}_i, \{C_{i,\ell}\}_{\mathsf{C} \in \mathcal{GU}}, C_i\big)$. Since $\mathcal{A}$ wins, we have that cert_i corresponds to an honest user. Also, by the fact that extraction for $\mathsf{NIZK}_{\mathsf{Dec}}$ fails, the signature opens to this user. But this means that the extractor of Hyb_3 extracted a witness for an honest user, which happens only with negligible probability.

C.3 Proof of Thm. 8

Proof. We show the property by a sequence of hybrids described next.

– Hyb_0 is the $\mathsf{t\text{-}Exp\text{-}UserAccountability}$ game.
– Hyb_1 is the same as Hyb_0 except that when the adversary queries $\mathsf{OMalJoin}$, we also run the extractor of $\mathsf{NIZK}_{\mathsf{ddh}}$ to extract the secrets sk_1, sk_2 used by $\mathcal{A}$. We output $\perp$ and abort if the extraction fails to output a valid witness.
– Hyb_2 is the same as Hyb_1 except that in the case of $\mathsf{OMalJoin}$ we also use the extractor of $\mathsf{NIZK}_{\mathsf{Enc}}$ and abort if the shares given by the adversary are inconsistent w.r.t the values $\mathsf{sk}_1, \mathsf{sk}_2$
– Hyb_3 is the same as Hyb_2 except that when the adversary outputs (m, π), we also run the extractor of $\mathsf{NIZK}_{\mathsf{PS}}$ to get the value sk used for the signature of knowledge.

Hyb_0 and Hyb_1 have outputs that are statistically close by the knowledge soundness property of $\mathsf{NIZK}_{\mathsf{ddh}}$. Similarly, this is the case for Hyb_1 and Hyb_2 under the knowledge soundness of $\mathsf{NIZK}_{\mathsf{PS}}$, and for Hyb_2 and Hyb_3 under the knowledge soundness of $\mathsf{NIZK}_{\mathsf{Enc}}$.

We next claim that Hyb_4 does not abort, that is, the extracted witness from the adversary during its challenge is the correct key of either an honest or a malicious user. This follows by the unforgeability of PS

signature scheme. We can build a reduction $\mathcal{B}$ that works as in the case of Exp-Unforge. Indeed, the only difference is the handling of OMalJoin. In this case, we use the value $\mathsf{sk} = \mathsf{sk}_1 + \mathsf{sk}_2$ extracted from $\mathsf{NIZK}_{\mathsf{ddh}}$ and make a signing query to the PS signing oracle. If the adversary forces Hyb_3 to abort, it must be the case that we extract a valid message/signature pair that we did not queried in the signing oracle.

To conclude the proof, it is enough to note that since the adversary behaves honestly (which is verified by the extracted witnesses in the previous hybrids) a valid (m, π) pair is always correctly de-anonymized.

C.4 Proof of Thm. 10

Proof. We show the two properties property by a sequence of hybrids described next. We show the case for t-Exp-UAnon&MalGuard and describe the changes needed for Exp-UAnon&MalGM.

- Hyb_0 is the t-Exp-UAnon&MalGuard game.
- Hyb_1 is the same as Hyb_0 with the following changes: when an honest user is registered through OHonJoin, we compute honestly the guardians shares, but we sample two random elements $(C_1, C_2) \leftarrow \mathbb{G}_2^2$ as the share given to the manager. We also sample uniform $T \leftarrow \mathbb{G}_T$. We simulate a proof π_{Enc} to convince the adversary about the correctness of the shares. We answer the oracle queries of the adversary as follows:
  - $\mathsf{OMemberAuth}(\mathsf{m}, \mathsf{cert}_i)$: We sample random $\sigma_1', \sigma_2' \leftarrow \mathbb{G}_1$ and simulate the proof of knowledge π_{PS}. We save the tuple $(\mathsf{m}, \mathsf{cert}_i, \sigma_1', \sigma_2')$ for future reference.
  - $\mathsf{ODeanon}(\mathsf{m}, \pi = \sigma_1', \sigma_2', \pi_{\mathsf{P}}, \mathsf{access})$: if there is an no saved entry containing $(\mathsf{m}, \mathsf{cert}_{i^*}, \sigma_1', \sigma_2')$ act honestly. Otherwise, parse $\mathsf{cert}_{i^*} = (\mathrm{ID}_{i^*}, \{C_{i^*,\ell}\}_{\mathsf{C}_\ell \in \mathcal{GU}}, C_{i^*})$, and $\mathsf{access} = \{(B_{i,\ell}, \pi_{i,\ell})\}_{i=1}^{n_U}$ and abort if there are no $t+1$ indices with correct proofs for all parties. If such indices exist, let $\left\{(B_{i,\ell_j}, \pi_{i,\ell_j})\right\}_{i=1,j=1}^{n_U,t+1}$ be these indices. For $i \neq i^*$ sample a random value B_i and for i^*, set $B_i = e(\sigma_2', h)e(\sigma_1', XY_1^{a_i^*})^{-1} \prod_{j=1^t+1} B_{i,j_k}^{-\lambda_j}$. Simulate the proofs of knowledge π_{Dec}.

  For the challenge query, we sign the message as in OMemberAuth. Signing is independent of the choice of b.

First, in the second hybrid, whenever $\mathcal{A}$ queries ODeanon on a valid signature, it can never be a signature that opens to an honest party not previously queried to OMemberAuth, otherwise $\mathcal{A}$ breaks user non-frameability. This means that all signatures queried are opened as in Hyb_0. Now, by ddh in group $\mathbb{G}_2$, the (encrypted) manager shares reveal no information to the adversary. It is enough to note that the messaged signed, $\mathsf{sk} = \mathsf{sk}_1 + \mathsf{sk}_2$, is a uniform message and the adversary can only decrypt and recover sk_1. Similarly, by ddh in $\mathbb{G}_1$, and zero knowledge of $\mathsf{NIZK}_{\mathsf{PS}}$ the signatures issued during queries to OMemberAuth or the challenge query are indistinguishable to honestly computed ones. Finally, because the challenge signature is independent of b, no adversary can win the latter game with non-negligible advantage.

For the case of Exp-UAnon&MalGM, we act respectively but we simulate the shares issued to the guardians. More concretely, we give random encryptions for the honest guardians and find values $P_1, \ldots, P_t$ that convince the adversary – this can be done by solving a linear system in the exponent as done in [8, Lemma 3]. To de-anonymize a signature, for user i^* we simply compute random values B_{i,ℓ_j} conditioned on $e(\sigma_2', h) = e(\sigma_1', XY_1^{a_i^*}) \prod_{j=1}^{t+1} B_{i,j_k}^{\lambda_j} e(\sigma_1', Y_0^{\mathsf{sk}_1})$ and simulate proofs of knowledge π_{Dec}. When the adversary de-anonymizes this message, it will open to ID^*. Arguing as before, in this case the first part of the secret key $\mathsf{sk} = \mathsf{sk}_1 + \mathsf{sk}_2$ is random since the adversary does not have enough shares to get this value and compare with the commitment given.

$\underline{\mathsf{JoinUser}}\langle U_i(\mathrm{ID}), \mathcal{KI}(\mathsf{sk}^{\mathcal{KI}}\mathsf{st}^{\mathcal{KI}}), \mathcal{GM}(\mathsf{sk}^{\mathcal{GM}}, \mathsf{st}^{\mathcal{GM}})\rangle(\mathsf{pp}, \mathsf{PK})$

$\mathcal{KI}$:

- Sample and send $a \leftarrow \mathbb{Z}_p, \hat{g} \leftarrow \mathbb{G}_1$.

U_i:

- Parse the key issuer's key $\mathsf{pk}^{\mathcal{KI}}$ as (X, Y_0, Y_1)
- Sample secret key shares $\mathsf{sk}_1, \mathsf{sk}_2 \leftarrow \mathbb{Z}_q^*$. Set $\mathsf{sk} = \mathsf{sk}_1 + \mathsf{sk}_2$.
- Set $\hat{g}_{\mathsf{sk}_1} := \hat{g}^{\mathsf{sk}_1}$; $g_{\mathsf{sk}_2} := g^{\mathsf{sk}_2}$; $\hat{g}_{\mathsf{sk}_2} := \hat{g}^{\mathsf{sk}_2}$.
- Compute $\pi_{\mathsf{DDH},1} \leftarrow \mathsf{NIZK}_{\mathsf{ddh}}.\mathsf{Prove}((g, g_{\mathsf{sk}_1}, \hat{g}, \hat{g}_{\mathsf{sk}_1}), \mathsf{sk}_1)$; $\pi_{\mathsf{DDH},2} \leftarrow \mathsf{NIZK}_{\mathsf{ddh}}.\mathsf{Prove}((g, g_{\mathsf{sk}_2}, \hat{g}, \hat{g}_{\mathsf{sk}_2}), \mathsf{sk}_2)$
- Share $Y_0^{\mathsf{sk}_1}$ to the guardians in a verifiable manner as follows:
  - Sample $p_1, \ldots, p_t \leftarrow \mathbb{Z}_q$. Let $P(X) = \mathsf{sk}_1 + \sum_{i=1}^{t} p_i X^i$.
  - For $i \in \{1, \ldots, t\}$ compute a commitment to the coefficients of P as $P_i := \hat{g}^{p_i}$.
  - For each $\mathsf{C}_\ell \in \mathcal{GU}$, compute encrypted shares $C_\ell := (C_{\ell,1}, C_{\ell,2}) := (h^r, Z_\ell^r Y_0^{P(\ell)})$, value $T_\ell = e(\hat{g}_{\mathsf{sk}_1} \cdot \prod_{i=1}^{t} P_i^{\ell^i}, Y_0)$, [a] and proof $\pi_{\mathsf{enc},\ell} \leftarrow \mathsf{NIZK}_{\mathsf{enc}}.\mathsf{Prove}((\hat{g}, h, Z_\ell, C_{\ell,1}, C_{\ell,2}, T_\ell), r))$.
- Verifiably secret share the value $Y_0^{\mathsf{sk}_2}$ to the manager as follows:
  - Compute encrypted share for the group manager $C := (C_1, C_2) := (h^r, Z^r Y_0^{\mathsf{sk}_2})$ and value $T = e(\hat{g}_{\mathsf{sk}_2}, Y_0)$.
  - Compute proof $\pi_{\mathsf{enc}} \leftarrow \mathsf{NIZK}_{\mathsf{enc}}.\mathsf{Prove}(\hat{g}, H, Z, C_1, C_2, T), r))$.
- Output $\left(\hat{g}_{\mathsf{sk}_1}, g_{\mathsf{sk}_2}, \hat{g}_{\mathsf{sk}_2}, \pi_{\mathsf{DDH},1}, \pi_{\mathsf{DDH},2}, \{P_\ell, C_\ell, \pi_{\mathsf{enc},\ell}\}_{\mathsf{C}_\ell \in \mathcal{GU}}\, C, \pi_{\mathsf{enc}}\right)$.

$\mathcal{KI}$:

- For $i \in \{1, \ldots, t\}$ compute $T = e(\hat{g}_{\mathsf{sk}_1} \cdot \prod_{i=1}^{t} P_i^{\ell^i}, Y_0)$. If $\mathsf{NIZK}_{\mathsf{enc}}.\mathsf{Vrfy}((\hat{g}, H, Z, C_1, C_2, T), \pi_{\mathsf{enc},\ell}) = 0$, abort.
- If $\mathsf{NIZK}_{\mathsf{enc}}.\mathsf{Vrfy}(\hat{g}, H, Z, C_1, C_2, T), \pi_{\mathsf{enc}})) = 0$, abort.
- If $\mathsf{NIZK}_{\mathsf{ddh}}.\mathsf{Vrfy}((g, g_{\mathsf{sk}_1}, \hat{g}, \hat{g}_{\mathsf{sk}_1}), \pi_{\mathsf{ddh},1}) = 0$ or $\mathsf{NIZK}_{\mathsf{ddh}}.\mathsf{Vrfy}((g, g_{\mathsf{sk}_2}, \hat{g}, \hat{g}_{\mathsf{sk}_2}), \pi_{\mathsf{ddh},1}) = 0$, abort.
- Blindly sign sk by computing $\sigma_2 = \hat{g}^{x+y_1 a}(\hat{g}_{\mathsf{sk}_1}\hat{g}_{\mathsf{sk}_2})^{y_0}$
- Send to all parties the user certificate $\mathsf{cert} := \left(\mathrm{ID}, a, \hat{g}, \{C_\ell\}_{\mathsf{C}_\ell \in \mathcal{GU}}, C\right)$ and σ_2 to U_i.

U_i:

- Set $\sigma_1 := \hat{g}$ and $\sigma =: (a, \sigma_1, \sigma_2)$.
- Output $\mathsf{sk}_i := (\mathsf{sk}, \sigma)$.

[a] Product of P^{ℓ^i} is just a polynomial evaluation of $p(\ell)$ in the exponent

Fig. 14. Joining Procedure.

$\underline{\mathsf{MemberAuth}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \mathsf{sk}_i =: (\mathsf{sk}, \sigma))$

- From PK parse $\mathsf{pk}^{\mathcal{KI}}$ as the PS signature scheme public key.
- Compute $\pi \leftarrow \mathsf{SoK}_{\mathsf{PS}}.\mathsf{Sign}(\mathsf{pk}^{\mathcal{KI}}, \mathsf{m}, (\mathsf{sk}, \sigma))$. Note that $\pi = (\sigma_1', \sigma_2', \pi_{\mathsf{PS}})$ where (σ_1', σ_2') is a re-randomization of the signature given by $\mathcal{KI}$ and π_{PS} a Sigma protocol proof.
- Output π.

$\underline{\mathsf{AuthVrfy}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi)$

- From PK parse $\mathsf{pk}^{\mathcal{KI}}$ as the PS signature scheme public key.
- Output 1 iff $\mathsf{SoK}_{\mathsf{PS}}.\mathsf{Vrfy}(\mathsf{pk}^{\mathcal{KI}}, \mathsf{m}, \pi) = 1$.

Fig. 15. Anonymous Membership Proof Generation and Verification Algorithms. $\mathsf{SoK}_{\mathcal{L}}$ the is the signature of knowledge derived from a Sigma protocol for a language $\mathcal{L}$.

$\underline{\mathsf{ReqDeanon}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{sk}^{\mathcal{GM}})$
$\mathcal{GM}$ simply signs the message it wants to de-anonymize.

- Output $\mathsf{req} \leftarrow \mathsf{S.Sign}_{\mathsf{sk}_{\mathsf{S}}^{\mathcal{GM}}}(\mathsf{m} \| \pi)$.

$\underline{\mathsf{JudgeReq}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{req})$
Verify signature of $\mathcal{GM}$.

- Output $b \leftarrow \mathsf{S.Vrfy}_{\mathsf{vk}_{\mathsf{S}}^{\mathcal{GM}}}(\mathsf{m} \| \pi, \mathsf{req})$

Fig. 16. De-anonymization Request Algorithms.

$\underline{\mathsf{GrantDeanon}}\left\langle \mathsf{C}_1(\mathsf{sk}_1^{\mathcal{GU}}), \ldots, \mathsf{C}_n(\mathsf{sk}_n^{\mathcal{GU}}) \right\rangle (\mathsf{pp}, \mathsf{PK}, \mathsf{req}, \mathsf{m}, \pi)$. Each $\mathsf{C}_\ell \in \mathcal{GU}$ proceeds as follows:

- If $\mathsf{JudgeReq}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{req}) = 0$, abort.
- Parse $\pi := (\sigma_1', \sigma_2', \pi_{\mathsf{PS}})$.
- Let n_U be the number of users at the time. For each $i \in [n_U]$ in some fixed (e.g., lexicographic w.r.t. ID) order
  - Let $\mathsf{cert}_i := \left(\mathrm{ID}_i, a_i, \hat{g}_i, \{C_{i,\ell} = (C_{i,\ell,1}, C_{i,\ell,2})\}_{\mathsf{C}_\ell \in \mathcal{GU}}, C_i = (C_{i,1}, C_{i,2})\right)$.
  - Compute $B_{i,\ell} = e(\sigma_1', C_{i,\ell,2} C_{i,\ell,1}^{-z_\ell})$.
  - Compute $\pi_{i,\ell} = \mathsf{NIZK}_{\mathsf{Dec}}.\mathsf{Prove}((\sigma_1', h, Z_\ell, C_{i,\ell,1}, C_{i,\ell,2}, B_{i,\ell}), z_\ell)$ where $(C_{i,\ell,1}, C_{i,\ell,2})$ is the share of $Y_0^{\mathsf{sk}_1}$ from user i to C_ℓ
- Output $\mathsf{access}_\ell := \{(B_{i,\ell}, \pi_{i,\ell})\}_{i=1}^{n_U}$.

$\underline{\mathsf{Deanon}}(C_{i,1}, C_{i,2}), \mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{sk}^{\mathcal{GM}})$.

- Parse $\pi := (\sigma_1', \sigma_2', \pi_{\mathsf{PS}})$.
- Parse $\mathsf{access}_\ell := \{(B_{i,\ell}, \pi_{i,\ell})\}_{i=1}^{n_U}$ for all ℓ.
- For $i \in [n_U]$:
  - Let $\mathsf{cert}_i := \left(\mathrm{ID}_i, a_i, \hat{g}_i, \{C_{i,\ell} = (C_{i,\ell,1}, C_{i,\ell,2})\}_{\mathsf{C}_\ell \in \mathcal{GU}}, C_i = (C_{i,1}, C_{i,2})\right)$.
  - Let $B_{i,j_1}, \ldots, B_{i,j_{t+1}}$ be $t+1$ values such that
$$\mathsf{NIZK}_{\mathsf{Dec}}.\mathsf{Vrfy}((\sigma_1', H, Z_{j_k}, C_{i,j_k,1}, C_{i,j_k,2}, B_{i,j_k}), \pi_{i,j_k}) = 1.$$
    and compute $\lambda_1, \ldots, \lambda_{t+1}$ the Lagrange coefficients corresponding to the points (j_k, B_{i,j_k}) for $1 \le k \le t+1$
  - Compute $B_i = e(\sigma_1', C_{i,2} C_{i,1}^{-z})$ and $\pi_i = \mathsf{NIZK}_{\mathsf{Dec}}.\mathsf{Prove}((\sigma_1', H, Z, C_i, C_i, B_i), z)$.
  - If $e(\sigma_2', h) = e(\sigma_1', X Y_1^{a_i}) B_i \prod_{j=k}^{t+1} B_{i,j_k}^{\lambda_j}$,
    - set $\mathsf{proof}_{\mathrm{ID}} = \left(\{B_{i,j_k}, \pi_{i,j_k}\}_{k=1}^{t+1}, B_i, \pi_i\right)$
    - output $(\mathrm{ID}_i, \mathsf{proof}_{\mathrm{ID}})$.
  - Else, continue to next i.
- If the check does not hold $\forall i \in n_U$ then output $\perp$.

$\underline{\mathsf{Judge}}(\mathsf{pp}, \mathsf{PK}, \mathsf{m}, \pi, \mathsf{access}, \mathsf{cert}_{\mathrm{ID}}, \mathsf{proof}_{\mathrm{ID}})$

- Parse $\mathsf{cert}_i := \left(\mathrm{ID}_i, a_i, \hat{g}_i, \{C_{i,\ell}\}_{\mathsf{C} \in \mathcal{GU}}, C_i\right)$.
- Parse $\pi := (\sigma_1', \sigma_2', \pi_{\mathsf{PS}})$.
- Parse $\mathsf{proof}_{\mathrm{ID}} = \left(\{B_{i,j_k}, \pi_{i,j_k}\}_{k=1}^{t+1}, B_i, \pi_i\right)$.
- Output 1 iff
  1. For all $1 \le k \le t+1$, $\mathsf{NIZK}_{\mathsf{Dec}}.\mathsf{Vrfy}((\sigma_1', H, Z_{j_k}, C_{i,j_k,1}, C_{i,j_k,2}, B_{i,j_k}), \pi_{i,j_k}) = 1$.
  2. $\mathsf{NIZK}_{\mathsf{Dec}}.\mathsf{Vrfy}((\sigma_1', H, Z, C_{i,1}, C_{i,2}, B_i), \pi_i) = 1$.

Fig. 17. De-anonymization Algorithms.

$\underline{\text{Hand-off}}\ \left\langle \mathsf{C}_\ell^{(i)}(\ell, s_{i,\ell}), \mathsf{C}_\ell^{(i+1)}(\ell) \right\rangle_{\mathsf{C}_\ell^{(i)} \in \mathcal{GU}^{(i)}, \mathsf{C}_\ell^{(i+1)} \in \mathcal{GU}^{(i+1)}} (\mathsf{pk})$.

- Party $\mathsf{C}_\ell^{(i)}$ holds a secret value-commitment pair $s_{i,\ell} \in \mathbb{Z}_p$, $g^{s_{i,\ell}} \in \mathbb{G}$. $[g^s]_t$ is public
- Parties of $\mathcal{GU}^{(i+1)}$ verifiably create two independent secret shares of a random value r, namely $[r]_t, [\tilde{r}]_t$, called *coupled sharings* as explained in [10, Sec. 4]. They publish $[g^r]_t, [g^{\tilde{r}}]_t$,
- Parties of $\mathcal{GU}^{(i)}$ reconstruct the secret $[s-r]_t = [s]_t - [r]_t$ and announce it.
- Party $\mathsf{C}_\ell^{(i+1)}$, holding $\tilde{r}_\ell$ computes $s_{i+1,\ell} = \tilde{r}_\ell + s - r$. Note, that since $\tilde{r} = r$, the values $s_{i+1,\ell}$ constitute a (different) share of s, that is $[\tilde{s}]_t = (s_{i+1,1}, \ldots, s_{i+1,n})$ for $s = \tilde{s}$.
- All parties update the public values $[g^{\tilde{s}}]_t = [g^{\tilde{r}}]_t \cdot g^{s-r}$.
- Parties of $\mathcal{GU}^{(i)}$ erase their local state, namely, the shares of $[s]_t$.

Fig. 18. Hand-off phase.

$\underline{\mathsf{Soundness}_{\mathsf{TE},\mathcal{A}}(1^\lambda)}$

- $\mathsf{pp}_{\mathsf{TE}} \leftarrow \mathsf{TE.Setup}(n, t, 1^\lambda)$
- $(\mathsf{m}, \mathsf{pk}_{\mathsf{TE}}) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{TE}})$
- $c \leftarrow \mathsf{TE.Enc}_{\mathsf{pk}_{\mathsf{TE}}}(\mathsf{m})$
- $(\mathsf{m}', \mathsf{proof}_{\mathsf{m}}) \leftarrow \mathcal{A}(\mathsf{st}_{\mathcal{A}})$
- Output 1 iff $\mathsf{TE.ValidDec}_{\mathsf{pk}_{\mathsf{TE}}}(\mathsf{m}', c, \mathsf{proof}_{\mathsf{m}'}) = 1$ and $\mathsf{m} \neq \mathsf{m}'$.

Fig. 19. $\mathsf{Soundness}_{\mathsf{TE},\mathcal{A}}$

$\underline{(t,n)\text{-}\mathsf{Ind}_{\mathsf{TE},\mathcal{A}}(1^\lambda)}$

- $\mathsf{pp}_{\mathsf{TE}} \leftarrow \mathsf{TE.Setup}\ (n,t,1^\lambda)$
- $\mathcal{A}$ controls a set M of parties chosen adaptively.
- $(\mathsf{pk}_{\mathsf{TE}}, (\mathsf{sk}_{\mathsf{TE}}^i)_{i=1}^n) \leftarrow \mathsf{TE.KeyGen}(\mathcal{A})\langle U_i(\cdot)\rangle_{i \notin M}(\mathsf{pp}_{\mathsf{TE}})$.
- $(\mathsf{m}_0, \mathsf{m}_1) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{TE}}, \mathsf{pk}_{\mathsf{TE}}, \mathsf{st}^{\mathcal{A}})$.
- $c_b \leftarrow \mathsf{TE.Enc}_{\mathsf{pk}_{\mathsf{TE}}}(\mathsf{m}_b)$ for uniform $b \leftarrow \{0,1\}$.
- $b' \leftarrow \mathcal{A}(c_b, \mathsf{st}^{\mathcal{A}})$.
- Output 1 iff $|M| \leq t$ and $b = b'$.

Fig. 20. $(t,n)\text{-}\mathsf{Ind}_{\mathsf{TE},\mathcal{A}}$