Short Threshold Dynamic Group Signatures

Jan Camenisch[1], Manu Drijvers[1], Anja Lehmann[2], Gregory Neven[1] and Patrick Towa[2,3]

[1] DFINITY*
[2] IBM Research – Zurich
[3] ENS and PSL Research University

Abstract. Traditional group signatures feature a single issuer who can add users to the group of signers and a single opening authority who can reveal the identity of the group member who computed a signature. Interestingly, despite being designed for privacy-preserving applications, they require strong trust in these central authorities who constitute single points of failure for critical security properties. To reduce the trust placed on authorities, we introduce dynamic group signatures which distribute the role of issuer and opener over several entities, and support t_I-out-of-n_I issuance and t_O-out-of-n_O opening. We first define threshold dynamic group signatures and formalize their security. We then give an efficient construction relying on the pairing-based Pointcheval–Sanders (PS) signature scheme (CT-RSA 2018), which yields very short group signatures of two first-group elements and three exponents. We also give a simpler variant of our scheme in which issuance requires the participation of all n_I issuers, but still supports t_O-out-of-n_O opening. It is based on a new multi-signature variant of the PS scheme which allows for efficient proofs of knowledge and is a result of independent interest. We prove our schemes secure in the random-oracle model under a non-interactive q-type of assumption.

1 Introduction

Group signatures [23] are a fundamental cryptographic primitive which allows members of a user group to anonymously sign on behalf of the group after interacting with an issuer. That is, anyone can verify that a signature was computed by a group member, but only a designated authority called opener can reveal the identity of the signer. Variants of group signatures are for instance used for privacy-preserving authentication of Trusted Platform Modules (TPMs) in user devices [30,17,34,28,18] and of vehicles in Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication [53,45,44].

The standard definition of group signatures places trust on a single issuer and a single opener. It means that a corrupt issuer can create credentials of which the signatures would open to no group member, and that there is no anonymity against a corrupt opener. For many applications, and in particular

* Work partially done while the corresponding authors were at IBM Research – Zurich.

for V2V communication, it is simply prohibitive to have a single authority able to issue untraceable credentials or to have a single opener able to trace all users. The standard solution in such cases is to use threshold cryptography [51,25] to distribute the capabilities of the issuer and opener over multiple entities, of whom a threshold number must collaborate to add a user or open a signature.

Many group signature schemes follow a modular "sign-and-encrypt-and-prove" approach [1,4,11], where a user's signing key is a certificate on her identity, and a group signature contains an encryption of her identity and a zero-knowledge proof (bound to the signed message) that the user knows a valid certificate for the encrypted identity.

The modular use of encryption to enable opening readily allows for threshold opening: it suffices to replace the underlying encryption scheme with another that supports threshold decryption [25], as Blömer et al. [9] pointed out. Nevertheless, lifting a group-signature scheme to a threshold-issuance setting is not as straightforward.

Short Group Signatures. The most efficient ("GetShorty") group signatures [8,46] depart from the sign-and-encrypt paradigm though, yielding the shortest signature sizes to date [41]. Instead of adding an encryption of the user's identity to every signature, they rely on the issuer for opening. To reveal the identity of the user that generated a signature, the issuer maintains a list of the membership credentials it has generated and tests the signature against each entry. It makes opening expensive for the benefit of having short signatures, which perfectly fits for all applications where signatures must be short and opening an uncommon practice, such as in V2X communication. A disadvantage of this GetShorty approach is that it merges the roles of issuer and opener into a single party that has to be trusted for anonymity and unforgeability.

Unfortunately, these schemes are difficult to map to a threshold setting. A first problem is that to trace the signatures of a user, her identifier generated during the issuance protocol is necessary. A second issue is that their underlying base signature schemes, namely Camenisch–Lysyanskaya [21] and Pointcheval-Sanders [46,47] signatures, are not a priori suitable to a multi-signer setting as needed to distribute issuance. Indeed, with those signature schemes, all signers would have to agree on a common randomness.

Contributions. In this paper we propose the first provably-secure group signatures that no longer require trust in single authorities for issuance and opening, but instead distribute their roles over several parties.

Security Model for Threshold Group Signatures. We start by formalizing threshold dynamic group signatures and define their security in the presence of multiple issuers and openers. Our model features a number n_I of issuers and a number n_O of openers separate from the issuers. Any quorum of $t_I + 1$ issuers can add users to the group, whereas no collusion of t_I issuers can generate a valid credential. Besides, any $t_O + 1$ openers can recover the identity of a signer, but anonymity is guaranteed against up to t_O corrupt openers.

Short Threshold Group Signatures. We then present an efficient, provably secure instantiation based on Pointcheval–Sanders (PS) signatures [47]. It shares ideas with the "GetShorty" approach and adapts them to a threshold setting. We show that the roles of issuer and opener can be separate even with this approach, as long as the openers can still access the opening information generated during issuance. Nevertheless, the openers do not partake in the issuance protocol and are the only parties who should be able to retrieve it. The challenge thus consists in making sure, during issuance, that the opening information is correct, and that the openers (and only them) can later retrieve it.

However, even with separate roles, opting for the most efficient approach slightly weakens the security expected from a threshold scheme: it guarantees anonymity if only some issuers are corrupt but not all. Likewise, it achieves traceability if only some openers are corrupt but not all.

The signatures of our scheme are short as they comprise only 2 first-group elements and 3 exponents. The computation and verification of our group signatures only costs a few exponentiations in the first group and pairing computations (see Table 1). They respectively consist in proving and verifying knowledge of a PS signature obtained from a threshold number of issuers. The size and computational efficiency of our threshold group signatures therefore make them suitable for practical privacy-preserving applications. We prove our construction secure in the random-oracle model under a non-interactive q-type of assumption.

Simpler Distributed Group Signatures and Multi-Signatures. We also present a variant of our scheme that requires the participation of all n_I issuers to add users to the group, but still caters for threshold opening. It is has the benefit of permitting the corruption of all issuers but one. It is based on a multi-signature variant of the PS scheme that we build and prove secure in the plain public-key model (i.e., the signers do not have to prove knowledge of their secret keys) under the same q-assumption. This PS multi-signature scheme constitutes a contribution of independent interest. Multi-signatures compress the signatures of multiple signers on the same message into a single compact signature and are for instance used to optimize consensus protocols in distributed ledgers and blockchains. Unlike existing multi-signature schemes [43,10,12,42], PS multi-signatures allow for efficient zero-knowledge proofs of signatures, making them an interesting tool for the design of privacy-enhancing cryptographic protocols.

Related Work. Soon after their introduction by Chaum and Van Heyst [23], several group-signature schemes were presented, but Bellare, Micciancio and Warinschi [4] were the first to formalize the security properties of static group signatures. Later on, Bellare, Shi and Zhang [7] gave formal security definitions for dynamic group signatures in which users can join the group at any time. Early schemes were based on the strong RSA assumption [1,20], but the focus later shifted to bilinear maps [11,14,21,8,37,46,24] for their better efficiency. Recently, with the possible advent of quantum computers, several group-signature schemes [36,38,16,39,13] have been proposed, but they remain inefficient compared to their pairing-based counterparts in terms of key or signature sizes, and

signing cost. Even the scheme of Ling et al. [39] is far from being as efficient as pairing-based schemes, although it is the first scheme with signature size independent of the group size. Moreover, none of the post-quantum schemes so far supports threshold issuance, and building a fully distributed post-quantum group signature scheme is still an open problem.

Ghadafi [33] and Blömer et al. [9] considered group signatures with threshold opening, but did not address the more challenging task of threshold issuance. Manulis [40] introduced democratic group signatures in which there is no group manager. All members must participate to add a user to the group, and any member can open all group signatures, i.e., there is no anonymity within the group. Zheng et al. [54] extended democratic group signatures to enforce that at least a threshold number of members must collaborate to open signatures. In a sense, the extension of democratic group signatures due to Zheng et al. can be viewed as group signature schemes with distributed issuance and threshold opening. However, in addition to the poor anonymity guarantees that they provide, democratic group signature schemes are not applicable to a dynamic setting in which members join the group at a high frequency since public keys must then be refreshed. Furthermore, the public keys and signatures of the constructions of Manulis and of Zheng et al. are linear in the group size, making them impractical.

In their "Coconut" paper [52], Sonnino et al. proposed an anonymous credential system with threshold issuance and selective disclosure of user attributes. Though their techniques to achieve threshold issuance are similar to ours, their solution does not consider the issue of threshold opening, and therefore leaves aside the difficulty of realizing both threshold issuance and threshold opening while having short signatures. Besides, the authors do not provide a security model to analyze the security of their scheme. They only informally state properties that a scheme with threshold issuance and selective disclosure should satisfy, and then argue that their scheme does.

Gennaro, Goldfeder and Ithurburn recently proposed [31] extensions of the BBS [11] and CL [21] group-signature schemes that support threshold issuance. To achieve threshold opening, since those schemes follow the sign-and-encrypt paradigm, the authors point out that it suffices to replace the underlying encryption scheme with a threshold one as did Ghadafi [33] and Blömer et al. [9]. Nonetheless, this paradigm results in large signatures as explained above. Furthermore, Gennaro et al. do not provide a security model for threshold group-signature schemes. For the BBS scheme, they give a simulation argument for their threshold issuance protocol. For the CL scheme, they give a game-based proof that an adversary controlling less than a threshold number of parties cannot issue new credentials. Without a model that takes into account all the other aspects of group signatures scheme, especially threshold opening, it is difficult to grasp the exact security guarantees of their threshold schemes.

2 Preliminaries

This section introduces the notations used throughout the paper, as well as the hardness assumptions and building blocks on which our constructions rely.

2.1 Notation

Vectors are denoted in bold font. For an integer $n \geq 1$, $[n]$ denotes the set $\{1, \ldots, n\}$. For an integer k, $\binom{[n]}{k}$ represents the set of subsets of $[n]$ of cardinality k. If $k \leq 0$ or $k > n$, then $\binom{[n]}{k} := \emptyset$. The notation $\binom{[n]}{\leq k}$ stands for the set of subsets of $[n]$ of cardinality no greater than k. Given a group $\mathbb{G}$ with neutral element $1_{\mathbb{G}}$, $\mathbb{G}^*$ denotes $\mathbb{G} \backslash \{1_{\mathbb{G}}\}$.

2.2 Pairing Groups

An asymmetric pairing group consists of a tuple $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$ such that p is a prime number, $\mathbb{G}$, $\tilde{\mathbb{G}}$ and $\mathbb{G}_T$ are p-order groups, and such that $e\colon \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_T$ is a pairing, i.e., an efficiently computable non-degenerate ($e \neq 1_{\mathbb{G}_T}$) bilinear map. Type-3 pairing groups are pairing groups for which there is no efficiently computable homomorphism from $\tilde{\mathbb{G}}$ to $\mathbb{G}$.

2.3 Hardness Assumptions

This section presents the assumptions needed to prove our constructions secure.

Pointcheval and Sanders introduced [47] a new non-interactive q-type of assumption that they called the Modified q-Strong Diffie–Hellman (q-MSDH-1) assumption. They proved that it holds in the generic bilinear group model.

Definition 1 (q-MSDH-1 Assumption). *Let G be a type-3 pairing-group generator. The q-MSDH-1 assumption over G is that for all $\lambda \in \mathbb{N}$, for all $\Gamma = \left(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e\right) \leftarrow \mathsf{G}(1^\lambda)$, given Γ, $g \in_R \mathbb{G}^*$, $\tilde{g} \in_R \tilde{\mathbb{G}}^*$, and two tuples $\left(g^{x^\ell}, \tilde{g}^{x^\ell}\right)_{\ell=0}^{q} \in (\mathbb{G} \times \tilde{\mathbb{G}})^{q+1}$ and $(g^a, \tilde{g}^a, \tilde{g}^{ax}) \in \mathbb{G} \times \tilde{\mathbb{G}}^2$ for $x, a \in_R \mathbb{Z}_p^*$, no efficient adversary can, with non-negligible probability, compute a tuple $\left(w, P, h^{1/x+w}, h^{a/P(x)}\right)$ with $h \in \mathbb{G}^*$, P a polynomial in $\mathbb{Z}_p[X]$ of degree at most q and $w \in \mathbb{Z}_p$ such that the polynomials $X + w$ and P are coprime.*

Definition 2 (Symmetric Discrete Logarithm Assumption). *Let G be a type-3 pairing-group group generator. The Symmetric Discrete Logarithm (SDL) assumption [8] over G is that for all $\lambda \in \mathbb{N}$, for all $\Gamma = \left(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e\right) \leftarrow \mathsf{G}(1^\lambda)$, $g \in_R G^*$, $\tilde{g} \in_R \tilde{\mathbb{G}}^*$, $x \in_R \mathbb{Z}_p^*$, given $(\Gamma, g, \tilde{g}, g^x, \tilde{g}^x)$ as an input, no efficient adversary can compute x with non-negligible probability.*

Note that given $(\Gamma, g, \tilde{g}, h, \tilde{h})$, one can always verify that it is a valid SDL tuple by testing the equality $e(g, \tilde{h}) = e(\tilde{g}, h)$. Notice also that the SDL assumption is implied by the q-MSDH-1 assumption (Definition 1).

Fuchsbauer and Orru introduced an analog of the Diffie–Hellman Knowledge of Exponent assumption [3] in an asymmetric setting called the Asymmetric Diffie–Hellman Knowledge-of-Exponent assumption [29]. It is primarily used in the context of subversion-resistant non-interactive witness-indistinguishable proofs.

Definition 3 (Asymmetric Diffie–Hellman Knowledge-of-Exponent Assumption). *The (first-group) Asymmetric Diffie–Hellman Knowledge-of-Exponent (ADH-KE) game, parametrized by $\lambda \in \mathbb{N}$, for a type-3 pairing-group generator G, an adversary $\mathcal{A}$ and an extractor Ext is defined as follows:*

- $\Gamma := (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e) \leftarrow \mathsf{G}(1^\lambda)$; $g \in_R \mathbb{G}^*$
- $(X, Y, Z) \leftarrow \mathcal{A}(\Gamma, g)$
- $s \leftarrow \mathsf{Ext}\,(\Gamma, g, X, Y, Z)$
- *return* $b \leftarrow \left(g^s \neq X \wedge g^s \neq Y \wedge Z = Y^{\mathrm{dlog}_g(X)}\right)$.

In other words, $\mathcal{A}$ wins the game if (g, X, Y, Z) is a Diffie–Hellman tuple, but algorithm Ext can extract neither $\mathrm{dlog}_g(X)$ nor $\mathrm{dlog}_g(Y)$.

The ADH-KE assumption over a type-3 pairing-group generator G is that there exists an efficient algorithm Ext such that for all efficient adversary $\mathcal{A}$ for the ADH-KE game, $\Pr[b = 1]$ is negligible in λ.

2.4 Signatures

A signature scheme consists of 4 algorithms: a setup algorithm $\mathsf{Setup}(1^\lambda) \to pp$, a key-generation algorithm $\mathsf{KG}(pp) \to (vk, sk)$, a signing algorithm $\mathsf{Sign}(sk, m) \to \sigma$ and a verification algorithm $\mathsf{Verf}(vk, m, \sigma) \to \{0, 1\}$.

2.5 Pointcheval–Sanders Signature Scheme

Pointcheval and Sanders [47] proposed an efficient signature scheme that allows to sign message blocks $(m_1, \ldots, m_k)$ at once, and also to efficiently prove knowledge of signatures in zero-knowledge. Given a type-3 pairing-group generator G and security parameter $\lambda \in \mathbb{N}$, the PS-signature scheme in a pairing-group $\Gamma = \left(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e\right) \leftarrow \mathsf{G}(1^\lambda)$ consists of the following algorithms.

$\mathsf{PS.Setup}(1^\lambda, \Gamma, k) \to pp$: generate $\tilde{g} \in \tilde{\mathbb{G}}^*$. Return $pp \leftarrow (\Gamma, \tilde{g}, k)$.

$\mathsf{PS.KG}(pp) \to (vk, sk)$: generate $x, y_1, \ldots, y_{k+1} \in_R \mathbb{Z}_p$, compute $\tilde{X} \leftarrow \tilde{g}^x$, $\tilde{Y}_j \leftarrow \tilde{g}^{y_j}$ for $j \in [k+1]$, and set and return $vk \leftarrow (\tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{k+1})$ and $sk \leftarrow (x, y_1, \ldots, y_{k+1})$.

$\mathsf{PS.Sign}\,(sk, (m_1, \ldots, m_k)) \to \sigma$: choose $h \in_R \mathbb{G}^*$, $m' \in_R \mathbb{Z}_p$ and return $\sigma \leftarrow \left(m', h, h^{x + \sum_{j=1}^{k} y_j m_j + y_{k+1} m'}\right)$.

$\mathsf{PS.Verf}\,(vk, (m_1, \ldots, m_k), \sigma) \to b$: parse σ as (m', σ_1, σ_2), verify that $\sigma_1 \neq 1_{\mathbb{G}}$ and that $e\left(\sigma_1, \tilde{X} \prod_{j=1}^{k} \tilde{Y}_j^{m_j} \tilde{Y}_{k+1}^{m'}\right) = e(\sigma_2, \tilde{g})$. If so, return 1, otherwise return 0.

Pointcheval and Sanders proved this scheme to be existential unforgeable under the q-MDSH-1 assumption [47] stated in Definition 1.

In the random oracle model, the scheme remains secure under the same assumption if m' is computed as $\mathcal{H}(m_1, \ldots, m_k)$ [47, Corollary 12]. Noticing that the verification algorithm does *not* verify any property on m', and in particular that $m' = \mathcal{H}(m_1, \ldots, m_k)$, the scheme still allows for efficient zero-knowledge proofs of knowledge if m' is computed as such.

Proving Knowledge of a PS Signature. An important feature of the PS signature scheme is that one can efficiently prove in zero-knowledge knowledge of a signature. In Section 4, we show how to apply similar ideas to threshold dynamic group signatures.

PS signatures allow to efficiently prove knowledge of signatures since the group elements in the signatures can be re-randomized, and since the verification of a signature does not require to check any hash relation between the extra exponent m' and the signed message block. Given a verification key vk as above, to prove knowledge of $(\boldsymbol{m}, \sigma = (m', \sigma_1, \sigma_2))$ such that $e\left(\sigma_1, \tilde{X} \prod_{i=1}^{k} \tilde{Y}_i^{m_i} \tilde{Y}_{k+1}^{m'}\right) = e(\sigma_2, \tilde{g})$,

- the prover parses σ as (m', σ_1, σ_2), generates $r, t \leftarrow_{\$} \mathbb{Z}_p^*$, computes $(\sigma_1', \sigma_2') \leftarrow (\sigma_1^r, (\sigma_2 \sigma_1^t)^r)$ and sends the latter couple to the verifier
- the verifier checks that $\sigma_1' \neq 1_{\mathbb{G}_1}$, and if so, the prover and the verifier engage in a Schnorr zero-knowledge proof of knowledge [49] of $(\boldsymbol{m}, m', t)$ such that $\prod_{i=1}^{k} e\left(\sigma_1', \tilde{Y}_i\right)^{m_i} e\left(\sigma_1', \tilde{Y}_{k+1}\right)^{m'} e(\sigma_1', \tilde{g})^t = e(\sigma_2', \tilde{g}) e(\sigma_1', \tilde{X})^{-1}$. Indeed, $(\boldsymbol{m}, m', t) \mapsto \prod_{i=1}^{k} e\left(\sigma_1', \tilde{Y}_i\right)^{m_i} e\left(\sigma_1', \tilde{Y}_{k+1}\right)^{m'} e(\sigma_1', \tilde{g})^t$ is a group homomorphism from $\mathbb{Z}_p^{k+2}$ to $\mathbb{G}_T$.

It is a proof of knowledge since the Schnorr proof is, and it is zero-knowledge as σ_1' and σ_2' can be simulated by generating random values and as the Schnorr proof is zero-knowledge. Moreover, since the protocol is public-coin, it can be turned into a non-interactive proof-system in the random oracle model [6] using the Fiat–Shamir heuristic [27]. This non-interactive version can be turned into a signature of knowledge [22] on a message m by including m to the hash computation.

3 Threshold Dynamic Group Signatures

In this section, we define threshold dynamic group signatures. Classical dynamic group signatures [7] allow users to join a group of signers at any time by interacting with an issuer, and then sign anonymously on behalf of the group. A

verifier is then assured that a valid signature stems from a group member but cannot infer any information about her identity. Only a dedicated authority, the opener, can recover the identity of a member who computed a valid signature.

In terms of security, group signatures should guarantee anonymity even in the presence of a corrupt issuer and unforgeability (also known as traceability) even when the opener is corrupt. Still, trust in each entity is necessary for the respective properties. It is even worse for schemes in which the roles of issuer and opener are assumed by the same party [14,19] who then has to be trusted for both anonymity and unforgeability. This holds in particular for the GetShorty-type of signatures [8,46] which yield the most efficient instantiations to date.

In this work, we distribute the capabilities of the issuer and the opener over several entities to prevent them from becoming single points of failure. To reflect the difference between issuer and opener, we introduce two thresholds: we define schemes with $n_I > 1$ issuers of whom $t_I + 1 \leq n_I$ are required to add users to the group. Similarly, there are $n_O > 1$ openers and at least $t_O + 1$ openers must collaborate to open a signature and reveal the signer's identity.

We start by defining the syntax of dynamic group signatures with threshold issuance and threshold opening. We then formalize the security requirements that can be expected from such schemes.

3.1 Syntax

Formally, a (t_I, t_O)-out-of-(n_I, n_O) DGS scheme, or $\binom{n_I, n_O}{t_I, t_O}$-DGS scheme, with identity space ID (assumed not to contain $\perp$) consists of the following algorithms:

$\mathsf{GSetup}(1^\lambda, n_I, n_O, t_I, t_O) \to pp$: on the input of a security parameter, a number n_I of issuers, a number n_O of openers and two integer threshold values, generates public parameters which are assumed to be an implicit input to all the other algorithms. Those parameters are also assumed to contain n_I, n_O, t_I and t_O. Moreover, each issuer is assigned a public, fixed index $i \in [n_I]$. Similarly, each opener is assigned a public index $i \in [n_O]$.

$\langle \{\mathsf{IKG}(pp, i)\}_{i=1}^{n_I} \rangle \to \langle \{(ipk, isk_i, st_i)\}_{i=1}^{n_I} \rangle$: a key-generation protocol between all n_I issuers. At the end of the protocol, the issuers agree on a public key, and each of them holds a secret key and a state. The issuer public key is used to add users to the group, and to compute and verify group signatures.

$\mathsf{OKG}(pp, i \in [n_O]) \to (opk_i, osk_i, \mathbf{reg}_i)$: a key-generation algorithm for the openers. It returns a public key opk_i assumed to contain i, a secret key osk_i and a register $\mathbf{reg}_i$ initially empty. The public key is needed to add users to the group, and to compute and verify signatures. The secret key and the register are needed to open signatures. The tuple $(opk_i)_{i=1}^{n_O}$ of all opener public keys is further denoted $\boldsymbol{opk}$.

The group public key gpk consists of ipk and $\boldsymbol{opk}$, i.e., $gpk \leftarrow (ipk, \boldsymbol{opk})$.

$\langle \mathsf{GJoin.U}(id, I, gpk) \rightleftharpoons$
$\{\mathsf{GJoin.I}(st_i, isk_i, id, I, gpk)\}_{i \in I} \rangle \to \langle \mathbf{gsk}[id]/\perp, st_i' \rangle$: is a protocol between

a user with identity id and $t_I + 1 =: |I|$ issuers. If $id \in st_i$ for any $i \in [n_I]$, then the ith issuer aborts the protocol. At the end of the protocol, the user algorithm returns a user group secret key $\mathbf{gsk}[id]$ (or $\perp$ if the protocol fails) and the state st_i of each issuer is updated.

$\mathsf{GSign}(gpk, \mathbf{gsk}[id], m) \to \sigma$: a probabilistic algorithm that computes a signature σ on a message m on behalf of the group.

$\mathsf{GVerf}(gpk, m, \sigma) \to b \in \{0, 1\}$: a deterministic algorithm that verifies a group signature σ on a message m w.r.t. to a group public key gpk.

$\langle \{\mathsf{GOpen}(\mathbf{reg}_i, osk_i, O, gpk, m, \sigma)\}_{i \in I} \rangle \to \langle \{id_i/\perp\}_{i \in O} \rangle$: is a protocol between $t_O + 1 =: |O|$ openers, at the end of which each algorithm returns the identity of the user who computed σ on m, or $\perp$ in case of failure.

Note that contrarily to the model of Bellare et al. [7], in the model above, each opener maintains a register separate from the state of the issuers. These registers are necessary to open signatures, in addition to the opener secret keys. Those registers should rather be thought as the registers in the model of Bichsel et al. [8].

Correctness. Correctness captures the property that all honest issuers must agree on the same group public key. A signature σ computed on a message m with the secret key of a group-member id should also be accepted by the verification algorithm. Lastly, by executing the opening protocol, any set of $t_O + 1$ openers should all return id. These properties should hold with overwhelming probability regardless of the order in which users are added to the group.

3.2 Security Model

The security requirements for threshold DGS schemes are similar to the conventional ones for dynamic group signatures with a single issuer and a single opener [7], but adapted to a threshold setting. Those requirements are *anonymity*, which guarantees that a group signature reveals no information about the member who computed it, and *traceability*, which expresses the unforgeability property of group signatures.

Essentially, no collusion of t_I issuers should be able to add users and no collusion of t_O openers should be able to open signatures. Our definitions are flexible in the sense that they can require an additional fraction of openers to be honest for traceability, and of issuers to be honest for anonymity. This allows for more efficient schemes that may need slightly stronger assumptions to prove their security.

Global Variables. In the security experiments for $\binom{n_I, n_O}{t_I, t_O}$-DGS schemes, the challenger maintains global variables which are accessible to the experiment oracles (defined hereunder). These variables are a group public gpk, a table of honest-user group secret keys $\mathbf{gsk}$ of size $|ID|$, and

- Q_{GJoin} a set of user identities id that have joined the group, whether honestly via a $\mathsf{GJoin.U}$ query or dishonestly via a $\mathsf{GJoin.I}_i$ query
- Q_{Corrupt} a set of user identities id either corrupt from the beginning via a $\mathsf{GJoin.I}_i$ query or of which the group secret key has been revealed
- Q_{GSign} a set of signing queries (id, m, σ) made by the adversary and the responses to those
- Q_{GOpen} a set of message–signature pairs (m, σ) for which the adversary has made an opening query.

The sets Q_{GJoin}, Q_{Corrupt}, Q_{GSign} and Q_{GOpen} are initially empty, and the entries of gpk and $\mathbf{gsk}$ are initially set to $\perp$.

Oracles. This sections describes the oracles in the security experiments for $\binom{n_I,n_O}{t_I,t_O}$-DGS schemes. The oracles have access to global variables priorly defined and maintained by the challenger in each security experiment. Whenever the adversary queries a protocol-algorithm (for joining or opening) oracle, a protocol execution is triggered with all the other honest parties on the same inputs, and the adversary plays the role of the dishonest parties (dishonest users, or dishonest issuers or openers). During these executions, the adversary controls the network, i.e., it can forward, delay, drop or modify the messages sent by the various parties. However, as in prior models [8], protocols can only be executed in sequential order, i.e., the adversary cannot start a protocol execution if all the prior ones have not terminated. In particular, the adversary cannot interleave messages between protocol executions or execute multiple sessions of the same protocol in parallel.

In the following description of the oracles, if a verification fails, the oracle returns $\perp$. It is implicitly assumed that id is always in ID. Given a set Q, the statement "adds x to Q" means that $Q \leftarrow Q \cup \{x\}$. The oracles in the security experiments are then

$\mathcal{O}.\mathsf{GJoin.U}(id, I)$: checks that $I \in \binom{[n_I]}{t_I+1}$. It adds id to Q_{GJoin}. It runs the user joining algorithm on (id, I, gpk). An execution of protocol GJoin is triggered and during it, the challenger plays the role of the (honest) user and of the honest issuers, and the adversary plays the role of the corrupt issuers. At the end of the protocol, if algorithm $\mathsf{GJoin.U}$ returns a key $\mathbf{gsk}[id]$, the challenger updates $\mathbf{gsk}$ accordingly.

$\mathcal{O}.\mathsf{GJoin.I}_i(id, I)$: (for each honest issuer i) checks that $i \in I \in \binom{[n_I]}{t_I+1}$. It adds id to Q_{GJoin} and Q_{Corrupt}. It runs the issuer joining algorithm on $(st_i, isk_i, id, I, gpk)$. An execution of protocol GJoin is triggered and during it, the adversary plays the role of the (corrupt) user and of the corrupt issuers. The challenger plays the role of the honest issuers.

$\mathcal{O}.\mathsf{GSign}(id, m)$: checks that $id \in Q_{\mathsf{GJoin}} \setminus Q_{\mathsf{Corrupt}}$. It computes $\sigma \leftarrow \mathsf{GSign}(gpk, \mathbf{gsk}[id], m)$. It adds (id, m, σ) to Q_{GSign} and returns σ.

$\mathcal{O}.\mathsf{GOpen}_i(O, m, \sigma)$: (for each honest opener i) checks that $i \in O \in \binom{[n_O]}{t_O+1}$. It adds (m, σ) to Q_{GOpen}. It runs the opening algorithm on $(\mathbf{reg}_i, osk_i, O, gpk, m, \sigma)$. A GOpen protocol execution is triggered and the adversary plays the

role of the corrupt openers, while the challenger plays that of the honest ones.

$\mathcal{O}.\mathsf{RevealU}(id)$: adds id to Q_{Corrupt} and returns $\mathbf{gsk}[id]$.

$\mathcal{O}.\mathsf{ReadReg}(i, id)$: returns $\mathbf{reg}_i[id]$.

$\mathcal{O}.\mathsf{WriteReg}(i, id, v)$: (for each honest opener i) sets $\mathbf{reg}_i[id] \leftarrow v$, i.e., it write value v on the register of the ith opener for user id.

Experiment $\mathbf{Exp}^{\mathrm{ano}-b}_{\mathsf{DGS},\lambda,n_I,n_O,t_I,t_O}(\mathcal{A})$:
$pp \leftarrow \mathsf{GSetup}(1^\lambda, n_I, n_O, t_I, t_O)$
$\langle st_{\mathcal{A}}, \{(ipk, isk_i, st_i)\}_{i>t_I^*} \rangle \leftarrow \langle \mathcal{A}(\mathsf{keygen}, pp), \{\mathsf{IKG}(pp, i)\}_{i>t_I^*} \rangle$
$\forall i > t_O, (opk_i, osk_i, \mathbf{reg}_i) \leftarrow \mathsf{OKG}(pp, i)$
$gpk \leftarrow (ipk, \boldsymbol{opk})$
$\mathcal{O} \leftarrow \left\{\mathsf{GJoin.U}, (\mathsf{GJoin.I}_i)_{i>t_I^*}, \mathsf{GSign}, (\mathsf{GOpen}_i)_{i>t_O}, \mathsf{RevealU}, \mathsf{WriteReg}\right\}$
$(st'_{\mathcal{A}}, id_0^*, id_1^*, m^*) \leftarrow \mathcal{A}^{\mathcal{O}(gpk, (\mathbf{reg}_i)_i, \mathbf{gsk}, \cdot)}(\mathsf{choose}, st_{\mathcal{A}}, (opk_i)_{i>t_O})$
$\sigma^* \leftarrow \mathsf{GSign}(gpk, \mathbf{gsk}[id_b^*], m^*)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}(gpk, \mathbf{reg}, \mathbf{gsk}, \cdot)}(st'_{\mathcal{A}}, \sigma^*)$
if $id_0^*, id_1^* \in Q_{\mathsf{GJoin}} \setminus Q_{\mathsf{Corrupt}}$ and $\mathbf{gsk}[id_0^*], \mathbf{gsk}[id_1^*] \neq \perp$ and $(m^*, \sigma^*) \notin Q_{\mathsf{GOpen}}$
    return b'
else
    return 0

Experiment $\mathbf{Exp}^{\mathrm{trace}}_{\mathsf{DGS},\lambda,n_I,n_O,t_I,t_O}(\mathcal{A})$:
$pp \leftarrow \mathsf{GSetup}(1^\lambda, n_I, n_O, t_I, t_O)$
$\langle st_{\mathcal{A}}, \{(ipk, isk_i, st_i)\}_{i>t_I} \rangle \leftarrow \langle \mathcal{A}(\mathsf{keygen}, pp), \{\mathsf{IKG}(pp, i)\}_{i>t_I} \rangle$
$\forall i > t_O^*, (opk_i, osk_i, \mathbf{reg}_i) \leftarrow \mathsf{OKG}(pp, i)$
$gpk \leftarrow (ipk, \boldsymbol{opk})$
$\mathcal{O} \leftarrow \left\{\mathsf{GJoin.U}, (\mathsf{GJoin.I}_i)_{i>t_I}, \mathsf{GSign}, (\mathsf{GOpen}_i)_{i>t_O^*}, \mathsf{RevealU}, \mathsf{ReadReg}\right\}$
$(O^*, m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}(gpk, (\mathbf{reg}_i)_i, \mathbf{gsk}, \cdot)}(\mathsf{forge}, st_{\mathcal{A}}, (opk_i, osk_i)_{i>t_O^*})$
if $O^* \notin \binom{\{t_O^*+1, \ldots, n_O\}}{t_O+1}$ then return 0
$\langle \{id_i^*\}_{i \in O^*} \rangle \leftarrow \langle \{\mathsf{GOpen}(\mathbf{reg}_i, osk_i, O^*, gpk, m^*, \sigma^*)\}_{i \in O^*} \rangle$
if $\mathsf{GVerf}(gpk, m^*, \sigma^*) = 1$ and (case 1 or case 2)
with
    case 1) opening failed i.e.,
        $\exists i \in O^*$: $id_i^* = \perp$ or $\exists i, j \in O^*$: $id_i^* \neq id_j^*$
    case 2) opening was "incorrect", i.e., setting $id^* \leftarrow id^*_{\max O^*}$,
        $id^* \notin Q_{\mathsf{GJoin}}$ or $(id^* \in Q_{\mathsf{GJoin}} \setminus Q_{\mathsf{Corrupt}}$ and $(id^*, m^*, \sigma^*) \notin Q_{\mathsf{GSign}})$
    return 1
else
    return 0

Fig. 1. Security Experiments for $\binom{n_I, n_O}{t_I, t_O}$-DGS Schemes.

Anonymity. Anonymity ensures that a group signature reveals no information about the identity of the member who computed it as long as at most t_O openers are corrupt and the signature has not been opened. We use the indistinguishability-based definition where the adversary chooses two honest users and a message. It receives a group signature computed with the key of either of them, and it must determine the signer's identity better than by guessing. The adversary is given access to an opening oracle which it can query on all but the challenge signature, capturing a CCA-2 type of anonymity [11]. Dynamic corruption of group members is allowed, i.e., a signer may initially be honest but later corrupt. However, anonymity is guaranteed only for fully honest users, i.e., there is no forward anonymity. See Figure 1 for the detailed experiment.

The classical notion of anonymity relies on the honesty of the opener, which we adjust to a threshold setting and allow the adversary to corrupt up to t_O out of n_O openers. Without loss of generality, corrupt entities are always assumed to be the first ones, i.e., openers $1, \ldots, t_O$ are controlled by the adversary and $t_O + 1, \ldots, n_O$ are run by the challenger.

Concerning the issuers, our notion is flexible. Ideally, in schemes where the issuer and the opener are distinct entities, the issuer can be fully malicious in the anonymity game. In a distributed setting, it would translate in corrupting all n_I issuers. However, enforcing the corruption of all issuers may exclude some efficient schemes, such as ours which follow the GetShorty approach. We therefore parametrize our anonymity definition to additionally limit the number of issuers that may be corrupt. In the experiment, the adversary corrupts t_I^* issuers, with t_I^* treated as a function of t_I. The strongest anonymity guarantees are achieved when $t_I^* = n_I$, in which case the adversary would output the issuer public key itself. Our efficient scheme presented in Section 4 realizes anonymity for $t_I^* = t_I < n_I/2$, i.e., the largest possible value for an interactive key-generation process to guarantee terminate (robustness).

Lastly, it is worth noting that w.r.t. generation, our model is stronger than that of Bellare et al. [7] in the sense that the keys of corrupt authorities are not assumed to be honestly generated.

Definition 4 (Anonymity). *A* $\binom{n_I,n_O}{t_I,t_O}$*-DGS scheme* DGS *is anonymous if for every efficient adversary* $\mathcal{A}$*, the advantage* $\mathbf{Adv}^{\mathrm{ano}}_{\mathsf{DGS},n_I,n_O,t_I,t_O,\mathcal{A}}(\lambda)$ *of* $\mathcal{A}$ *defined as* $\left|\Pr[\mathbf{Exp}^{\mathrm{ano}-0}_{\mathsf{DGS},\lambda,n_I,n_O,t_I,t_O}(\mathcal{A}) = 1] - \Pr[\mathbf{Exp}^{\mathrm{ano}-1}_{\mathsf{DGS},\lambda,n_I,n_O,t_I,t_O}(\mathcal{A}) = 1]\right|$ *is negligible in* λ*.*

Traceability. This notion captures the unforgeability property expected from dynamic group signatures and guarantees that only users who have joined the group can compute valid group signatures. With single authorities, the opener can be corrupt but the issuer must be honest. Therefore, adapted to our threshold setting, up to t_I out of n_I issuers can be corrupt.

Traceability is then formalized through the opening capabilities of group signatures as described in Figure 1. It guarantees that for any valid signature σ on a message m, opening can neither fail (Case 1) nor reveal an "incorrect"

identity (Case 2). The first case means that an opener cannot identify any signer or that the openers do not agree on the identity of the signer. The second case means that the revealed identity has either never joined the group of signers, or has joined and is honest, but never signed m. The latter is sometimes formalized through a dedicated *non-frameability* requirement and we discuss our choice of combining both notions below.

Similarly to the case of anonymity, we parametrize the number of openers that the adversary can additionally corrupt via a bound t_O^*. The strongest traceability notion is achieved when $t_O^* = n_O$, i.e., when all openers can be corrupted. This is however not achievable when openers maintain state critical for opening, since the winning condition depends on a correct execution of protocol GOpen. In case of stateful opening, this requires the non-corrupt registers of at least $t_O + 1$ openers. Therefore, in such settings, at most $t_O^* = n_O - t_O - 1$ openers can be corrupt.

Definition 5 (Traceability). *A* $\binom{n_I, n_O}{t_I, t_O}$*-DGS scheme* DGS *is traceable if for all efficient adversary* $\mathcal{A}$*,* $\Pr\left[\mathbf{Exp}^{\text{trace}}_{\mathsf{DGS},\lambda,n_I,n_O,t_I,t_O}(\mathcal{A}) = 1\right]$ *is negligible in* λ*.*

On Non-Frameability. Classical definitions of group signatures with single authorities also include the notion of non-frameability. It reflects the idea that even if the issuer and opener are corrupt, they cannot falsely claim that an honest user computed a given valid signature. Since the opening algorithm in those definitions returns a long-term user public key, *in practice*, a public-key infrastructure would have to bind those keys to real-world identities; and such an infrastructure would be built with one or several certification authorities. However, if these certification authorities collude with the issuer and the opener, they would be able to frame an honest user. In other words, in real-world applications, users still need to trust some certification authority to protect them from malicious group-signature authorities even though the goal of non-frameability is precisely to avoid trust assumptions.

On the other hand, the rationale of threshold cryptography is that if there are many parties, some of them might in practice be corrupt but not all. Since we now consider group-signature schemes with several issuers and openers, we require that if less than respective threshold numbers of them are corrupt, no efficient adversary can forge a signature and falsely claim that an honest user computed it. It is precisely this requirement that is captured by the last winning condition of the above definition of traceability. Our scheme in Section 4 satisfies this property, but would not satisfy a definition in which all group-signature authorities are corrupt.

4 Our Threshold Dynamic Group Signatures

In this section, we build a threshold DGS scheme from (a variant of) PS signatures. We adopt the GetShorty approach of Bichsel et al. [8] instead of the traditional sign-and-encrypt paradigm. This approach avoids the extra encryption

of user identities and enables schemes with highly compact signatures despite supporting signature opening. Our resulting signatures are short, and computing and verifying them only require a few exponentiations in the first group and some pairing computations (see Table 1).

The efficiency of the GetShorty scheme of Bichsel et al. [8] comes at the price of fully trusted authority responsible for both issuance and opening. A threshold stetting allows to preserve the efficiency of the GetShorty approach, yet avoid the need for a single trusted entity. Our scheme shows that even with the GetShorty approach, the role of issuer and opener can be separated and distributed, and it enables t_I-out-of-n_I issuance and t_O-out-of-n_O opening. We still pay a small price for the efficiency, as not all issuers can be corrupt for anonymity and, likewise, not all openers can be malicious for traceability (contrarily to what might be expected). Still, moving to a threshold setting already avoids the most critical assumption, namely a fully trusted party, and instead tolerates corruption of some of them.

One challenge in designing our scheme is to separate the role of issuers and openers. It is necessary in the scheme of Bichsel et al. as the information needed for opening is created during issuance. Our scheme avoids that by assuming a public ledger to which users can upload their opening information during isssuance. This information is encrypted under the opener keys during issuance, and a user must prove to the issuers that she uploaded a valid ciphertext. We also present a scheme in Appendix C that does not assume a ledger but instead combines the roles of issuer and opener anew, and supports threshold issuance and opening.

We first define the variant of the PS signature scheme on which our scheme is based and then describe our threshold group signatures.

4.1 Variant of the PS Signature Scheme

Consider the PS signature scheme (Section 2.5) in which the extra scalar m' is computed as $\mathcal{H}(m)$. In the same vein, the group element h could also be returned by the hash function, i.e., $(m', h) \leftarrow \mathcal{H}(m)$. This would allow several signers of the same message to agree on a common base h. The scheme remains unforgeable and the main difference from the unforgeability proof of Pointcheval and Sanders [47, Section 4.3] is that when $\mathcal{H}$ is queried on a message $\boldsymbol{m}$ different from the challenge message, the reduction algorithm already prepares (σ_1, σ_2) to be later returned in case the adversary makes a signing query on $\boldsymbol{m}$. For more detail, see the unforgeability proof of the PS multi-signature scheme (introduced in Section 5.1) which relies on the same idea.

This technique is similar to that of Sonnino et al. [52] for their credential system. They hash a commitment to the signed message to obtain a base h, even though they apply it to the CT-RSA'16 version of the PS scheme (so without m') and do not formally prove that the scheme remains secure.

Moreover, assume that the messages to be signed are publicly indexed, i.e., that for every message $\boldsymbol{m}$ there exists a unique value $idx_{\boldsymbol{m}}$ known to any signer. To sign a message $\boldsymbol{m}$, instead of hashing to determine a scalar m' and a base

h, a signer could instead compute m' as $\mathcal{H}(idx_{\boldsymbol{m}})$. If the number of messages to be signed is known in advance to be polynomial, the scheme remains unforgeable under the same assumption since indexing messages to determine (m', h) is equivalent to specifying in the public parameters a pair (m', h) for each message $\boldsymbol{m}$. It is this variant of the scheme that is considered in our construction of threshold group signatures in Section 4.2. Therein, the messages signed are user secret keys sk_{id} indexed by the user identities id.

4.2 Construction with Separate Issuers and Openers

We first explain the main ideas of our construction, and then detail the protocols and algorithms.

Key Generation. During set-up, the issuers run the distributed key-generation protocol of Gennaro et al. [32] with t_I as a threshold to generate a public key for the PS signature scheme so that each of them holds a share of the secret key. The protocol guarantees that if $t_I < n_I/2$, then the protocol terminates (which cannot be enforced with a dishonest majority) and no colluding t_I issuers can infer any information about the secret key, whereas any $t_I + 1$ issuers can reconstruct it.

As for the openers, each of them simply generates a pair of ElGamal keys.

Join. For $t_I + 1$ issuers to add a user to the group, they all blindly sign with their PS secret-key share a random secret key sk_{id} chosen by the user. To do so, they need to agree on a common PS base h, so we used the variant from Section 4.1 of the PS signature scheme. The user group secret key consists of sk_{id} and the PS signature on it.

For each opener, the user encrypts a t_O-out-of-n_O Shamir share of sk_{id} and proves that the ciphertexts are correctly computed. With the ElGamal public keys of the openers, the issuers can verify the proofs and thus be convinced that the any $t_O + 1$ openers will later be able to retrieve correct shares of the user secret key if they can access the ciphertexts.

To make sure that these shares can later be retrieved by the openers, we assume the existence of an append-only ledger L accessible to all users, issuers and openers. Once the user has encrypted the shares of her secret key and has proved that she did so, she writes the encrypted shares and the proofs on the ledger. The issuers send their PS signature-shares only after verifying the proofs.

Therefore, each opener can later retrieve his shares of all group-member secret keys from the ledger. Note that since honest issuers add a given user identity id to the group only once and when the proofs are correct, there is only one entry with valid proofs per user that can open her signatures. This entry is the one denoted $L[id]$ in the descripiton of the opening algorithm.

Sign & Verify. To compute a group signature on a message m, the user computes a signature of knowledge on m of a valid PS signature on a user secret key sk_{id}. Verifying a signature on a message simply consists in verifying the signature of knowledge on it.

Open. To open a signature, any $t_O + 1$ openers first retrieve from the ledger their shares of user secret keys and store them in their registers. Once the openers' registers are updated, they test the signature to be opened against each entry in their registers until they find a user for which the shares match. It makes opening expensive for the benefit of having short signatures, which perfectly fits most practical scenarios, in which signatures should be short and opening an uncommon practice performed by resourceful authorities.

Scheme Description. To formally define the construction, let $\mathcal{H}_0 \colon \{0,1\}^* \to \mathbb{Z}_p \times \mathbb{G}^*$, and $\mathcal{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ be two random oracles (to compute non-interactive proofs of knownledge via the Fiat–Shamir heuristic [27]). Let also $ID \subseteq \mathbb{Z}_p$ denote a user identity space. The construction requires as building blocks

- the Section-4.1 variant of the PS signature scheme to sign user secret keys. It is further denoted PS
- an append-only ledger L with user identities as keys that is available to all users, issuers and openers
- secure (i.e., authenticated and confidential) channels
- a broadcast channel, i.e., a protocol between several parties that allows a sender to distribute a value to all the other parties so that the following three properties are satisfied:
  1. (termination) the protocol terminates
  2. (consistency) all honest parties receive the same value and
  3. (validity) if the sender is honest, then the value received by all honest parties is that of the sender.

Given a type-3 pairing group generator G and a security parameter $\lambda \in \mathbb{N}$, Our $\binom{n_I,n_O}{t_I,t_O}$-DGS scheme $\mathsf{PS\text{-}DGS}$ in a pairing group $\Gamma = \left(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e\right) \leftarrow \mathsf{G}(1^\lambda)$ is the following:

$\mathsf{GSetup}(1^\lambda, n_I, n_O, t_I, t_O) \to pp$: generate $(g, \tilde{g}) \in_R \mathbb{G}^* \times \tilde{\mathbb{G}}^*$. Set $pp_{\mathsf{PS}} \leftarrow (\Gamma, \tilde{g}, 1)$ and return $pp \leftarrow (pp_{\mathsf{PS}}, g, n_I, n_O, t_I, t_O)$.

$\langle \{\mathsf{IKG}(pp, i)\}_{i=1}^{n_I} \rangle \to \langle \{(ipk, isk_i, st_i)\}_{i=1}^{n_I} \rangle$: is a protocol between all the issuers who proceed as follows

- the issuers run three times the distributed key-generation protocol of Gennaro et al. [32] with t_I as a threshold in group $\tilde{\mathbb{G}}$ to obtain three uniformly distributed public values $\tilde{X}$, $\tilde{Y}_0$ and $\tilde{Y}_1$. At the end of the protocol, each issuer $i \in [n_I]$ holds shares x_i, $y_{0,i}$ and $y_{1,i}$ such that for any $I \in \binom{[n_I]}{t_I+1}$, if w_i denotes the Lagrange coefficient of issuer i, then $x \coloneqq \mathrm{dlog}_{\tilde{g}} \tilde{X} = \sum_{i \in I} x_i w_i$, and similarly for $y_0 \coloneqq \mathrm{dlog}_{\tilde{g}} \tilde{Y}_0$ and $y_1 \coloneqq \mathrm{dlog}_{\tilde{g}} \tilde{Y}_1$.
- issuer $i \in [n_I]$ returns $\left(ipk \leftarrow \left(\tilde{X}, \tilde{Y}_0, \tilde{Y}_1\right), isk_i \leftarrow (i, x_i, y_{0,i}, y_{1,i}),\ st_I \leftarrow \bot\right)$. The issuers send ipk to a certification authority which is assumed to make it publicly available so that anyone can get an authentic copy of it.

$\mathsf{OKG}(pp, i \in [n_O]) \to (opk_i, osk_i, \mathbf{reg}_i)$: generate an ElGamal pair of keys $(\tilde{f}_i \leftarrow \tilde{g}^{z_i}, z_i) \in \tilde{\mathbb{G}}^* \times \mathbb{Z}_p^*$ and initialize an empty register $\mathbf{reg}_i$. Set $opk_i \leftarrow (i, \tilde{f}_i)$ and $osk_i \leftarrow (i, z_i)$. Return $(opk_i, osk_i, \mathbf{reg}_i)$. Opener i sends opk_i to a certification authority.

The group public key gpk is set to $(ipk, \boldsymbol{opk})$.

GJoin : Assume that there is a broadcast channel between a user $\mathcal{U}$ and the t_I+1 issuers $\mathcal{I}_i$ in $I \in \binom{[n_I]}{t_I+1}$. Assume also that there is a secure channel between $\mathcal{U}$ and every issuer $\mathcal{I}_i$. In particular, this implies that an adversary cannot modify the messages sent by the user to the issuers: it can only forward, delay or drop them. Throughout the following description of the protocol, whenever an algorithm receives an abort or an ill-formed message, or when a verification fails, it interrupts the protocol execution by broadcasting an abort message to all participants and returning $\perp$. The joining protocol between the user and the issuers is as follows:

1. $\mathsf{GJoin.U}$, on input $\Big(id, I, gpk = \Big(ipk = (\tilde{X}, \tilde{Y}_0, \tilde{Y}_1), \boldsymbol{opk}\Big)\Big)$,
   - choose $sk_{id} \in_R \mathbb{Z}_p^*$
   - $(a', h) \leftarrow \mathcal{H}_0(id)$
   - $h_{sk} \leftarrow h^{sk_{id}}$; $g_{sk} \leftarrow g^{sk_{id}}$. Therefore, (g, h, g_{sk}, h_{sk}) is a DDH tuple. It helps the reduction algorithm of the traceability proof to efficiently extract the secret keys of adversarial users (under the ADH-KE assumption).
   - $\pi \leftarrow \mathsf{NIZK.Prove}\{sk_{id} : h_{sk} = h^{sk_{id}} \wedge g_{sk} = g^{sk_{id}}\}$
   - generate $p_1, \ldots, p_{t_O} \in_R \mathbb{Z}_p$ and set $P \leftarrow sk_{id} + \sum_{\ell=1}^{t_O} p_\ell X^\ell \in \mathbb{Z}_p[X]$
   - for $i \in [n_O]$, compute $s_i \leftarrow P(i)$, i.e., Shamir shares of sk_{id} for each opener
   - for $\ell \in [t_O]$, compute $h_\ell \leftarrow h^{p_\ell}$, i.e., verification values as in the Feldman verifiable secret sharing scheme [26]
   - for all $i \in [n_O]$:
     * $r_i \leftarrow_\$ \mathbb{Z}_p$
     * $\tilde{C}_i := (\tilde{C}_{i,0}, \tilde{C}_{i,1}) \leftarrow \left(\tilde{g}^{r_i}, \tilde{f}_i^{r_i} \tilde{Y}_0^{s_i}\right)$
     * $\pi_i \leftarrow \mathsf{NIZK.Prove}\left\{r_i : \tilde{C}_{i,0} = \tilde{g}^{r_i}, e\left(h, \tilde{C}_{i,1}/\tilde{f}_i^{r_i}\right) = e\left(h_{sk} \prod_{\ell=1}^{t_O} h_\ell^{i^\ell}, \tilde{Y}_0\right)\right\}$, i.e., compute a proof that $\tilde{C}_i$ encrypts the ith share of sk_{id}
   - set $L[id] \leftarrow \left(g_{sk}, h_{sk}, h_1, \ldots, h_{t_O}, \pi, \left(\tilde{C}_i, \pi_i\right)_{i \in [n_O]}\right)$
   - broadcast $\mathsf{written}$ to all $\mathcal{I}_i$

2. $\mathsf{GJoin.I}$, for $i \in I$, on input $(st_i, isk_i = (i, x_i, y_{0,i}, y_{1,i}), id, I, gpk)$
   - abort if $id \in st_i$
   - upon receiving $\mathsf{written}$ from $\mathcal{U}$:
     * $(a', h) \leftarrow \mathcal{H}_0(id)$
     * parse $L[id]$ as $\left(g_{sk}, h_{sk}, h_1, \ldots, h_{t_O}, \pi, \left(\tilde{C}_i, \pi_i\right)_{i \in [n_O]}\right)$
     * $\mathsf{NIZK.Verf}(g, h, g_{sk}, h_{sk}, \pi) \stackrel{?}{=} 1$, i.e., verify that it is a DDH tuple

* for $j \in [n_O]$, $\mathsf{NIZK.Verf}\left(h, (h_\ell)_{\ell=1}^{t_O}, \tilde{Y}_0, \tilde{f}_j, \tilde{C}_j, \pi_j\right) \stackrel{?}{=} 1$, i.e., verify that the ciphertexts encrypt correct shares for each opener
* $\Sigma_{i,2} \leftarrow h^{x_i + y_{i,1} a'} h_{sk}^{y_{i,0}}$ (i.e., blindly sign sk_{id} via h_{sk})
* $st_i \leftarrow st_i \cup \{id\}$
* send $\Sigma_{i,2}$ to $\mathcal{U}$ over a secure channel

3. $\mathsf{GJoin.U}$, upon receiving $\Sigma_{i,2}$ from all $\mathcal{I}_i$ for $i \in I$,
– $\Sigma \leftarrow \left(a', h, \prod_{i \in I} \Sigma_{i,2}^{w_i} = h^{x + y_0 sk_{id} + y_1 a'}\right)$, i.e., reconstruct the PS signature w.r.t. to ipk
– $\mathsf{PS.Verf}(ipk, sk_{id}, \Sigma) \stackrel{?}{=} 1$
– return $\mathbf{gsk}[id] \leftarrow (sk_{id}, \Sigma)$

$\mathsf{GSign}(ipk, \mathbf{gsk}[id], m) \to \sigma$: parse $\mathbf{gsk}[id] = (sk_{id}, \Sigma = (a', \Sigma_1, \Sigma_2))$. Generate $r \in_R \mathbb{Z}_p^*$. Compute $(\Sigma_1', \Sigma_2') \leftarrow (\Sigma_1^r, \Sigma_2^r)$ and

$$\pi \leftarrow \mathsf{NIZK.Prove}\{(sk_{id}, a') \colon \mathsf{PS.Verf}(ipk, sk_{id}, (a', \Sigma_1', \Sigma_2')) = 1\}(m).$$

That is, compute, with $\mathcal{H}_1$ as random oracle, a Schnorr signature of knowledge $\pi = (c, v_{sk}, v_{a'}) \in \mathbb{Z}_p^3$ on m of a pair (sk_{id}, a') such that $e\left(\Sigma_1'^{\,sk_{id}}, \tilde{Y}_0\right) e\left(\Sigma_1'^{\,a'}, \tilde{Y}_1\right) = e(\Sigma_2', \tilde{g}) e(\Sigma_1', \tilde{X})^{-1}$. Return $\sigma \leftarrow (\Sigma_1', \Sigma_2', \pi)$.

$\mathsf{GVerf}(ipk, m, \sigma) \to b \in \{0, 1\}$: parse σ as $(\Sigma_1, \Sigma_2, \pi)$. Return $\mathsf{NIZK.Verf}(gpk, \Sigma_1, \Sigma_2, m, \pi)$. That is, return 1 if

$$c = \mathcal{H}_1\left(ipk, \Sigma_1', \Sigma_2', e\left(\Sigma_1'^{\,v_{sk}}, \tilde{Y}_0\right) e\left(\Sigma_1'^{\,v_{a'}}, \tilde{Y}_1\right) e\left(\Sigma_2'^{\,c}, \tilde{g}\right) e\left(\Sigma_1'^{\,-c}, \tilde{X}\right), m\right)$$

and 0 otherwise.

GOpen : is run by $t_O + 1$ openers $\mathcal{O}_i$ (for $i \in O$) to recover the identity of the user who computed a valid group signature $\sigma = (\Sigma_1, \Sigma_2, \pi)$ on a message m. To do so, the openers first update their registers by checking the public ledger L. Then, the opening algorithms loop over the entries of their registers $\mathbf{reg}_i$ containing encryptions of shares $\tilde{Y}_0^{s_i}$ recorded during executions of protocol GJoin. For each identity id for which they have a share, the opening algorithms use their shares to determine whether (a', Σ_1, Σ_2) (with $(a', h) = \mathcal{H}_0(id)$) is a valid PS signature on the unique value determined by their t_O+1 shares of the secret key of user id. If it is the case, the algorithms return id. If no such identity is found, the opening algorithm returns $\perp$. The protocol assumes a broadcast channel between the participating openers, and also that the protocol is aborted as soon as an algorithm receives an abort or an ill-formed message.
Formally, $\mathsf{GOpen}\left(\mathbf{reg}_i, osk_i = (i, z_i), I, gpk, m, \sigma = (\Sigma_1, \Sigma_2, \pi)\right)$ proceeds as follows:
1. if $\mathsf{GVerf}(ipk, m, \sigma) = 0$ then return $\perp$

2. for all id such that $L[id] \neq \perp$, if $\mathbf{reg}_i[id] = \perp$ then parse $L[id]$ as $\left(g_{sk}, h_{sk}, h_1, \ldots, h_{t_O}, \pi, (\tilde{C}_i, \pi_i)_{i\in[n_O]}\right)$ and set $\mathbf{reg}_i[id] \leftarrow \tilde{C}_{i,1}/\tilde{C}_{i,0}^{z_i}$
3. for all id such that $\mathbf{reg}_i[id] \neq \perp$, compute $T_{id,i} \leftarrow e\left(\Sigma_1, \tilde{C}_{i,1}/\tilde{C}_{i,0}^{z_i}\right)$
4. broadcast $S_i \leftarrow \{(id, T_{id,i})\}_{id\colon \mathbf{reg}_i[id]\neq\perp}$ to all the openers in I
5. upon receiving S_j from all the other openers $\mathcal{O}_j$ (for $j \in I\backslash\{i\}$),
   - for $j \in I$, compute $w_j \leftarrow \prod_{\ell\in I\backslash\{j\}} \ell/(\ell - j)$, i.e., the jth Lagrange coefficient
   - for all $(id, T_{id,i}) \in S_i$ (in lexicographic order of user identities)
     * if $\exists j \in I\colon (id, *) \notin S_j$ then continue
     * $(a', h) \leftarrow \mathcal{H}_0(id)$
     * for all $j \in I\backslash\{i\}$, retrieve $(id, T_{id,j})$ from S_j
     * if $e\left(\Sigma_1, \tilde{X}\,\tilde{Y}_1^{a'}\right)\prod_{j\in I} T_{id,j}^{w_j} = e(\Sigma_2, \tilde{g})$ then return id
6. return $\perp$ (i.e., in case the previous equality holds for no tuple in $\mathbf{reg}_i$).

Remark 1. Our signing and verification algorithms only need the short issuer public key, not the entire group public key.

Theorem 1 (Correctness). PS-DGS *is correct.*

Proof. The correctness of the scheme follows from the correctness of the key-generation protocol of Gennaro et al., the correctness of the Shamir sharing scheme, the correctness of the Elgamal encryption scheme (relevant during issuance and opening) and the completeness of Schnorr proofs. □

The lemmas below give important arguments for the proofs of anonymity and traceability. The complete proofs of the following lemmas and theorems are deferred to Appendix B.

Lemma 1 (Unanimity). *Protocol* GOpen *is unanimous between honest openers in the anonymity and traceability security experiments. That is, for all* (O, m, σ)*, if* id_i *denotes the output of the* i*th opening algorithm and* $\mathcal{HO}$ *the index set of honest openers, then* $id_i = id_j$ *for all* $i, j \in O \cap \mathcal{HO}$*.*

Lemma 2 (Forgery). *If the SDL assumption over the pairing-group generator* G *holds, then in the anonymity and traceability security experiments for* PS-DGS*, no efficient adversary* $\mathcal{A}$ *can forge, with non-negligible probability, a signature that opens to an honest user for which it has not queried the secret keys. That is, no efficient adversary* $\mathcal{A}$ *can, with non-negligible probability, make an oracle query* $\mathsf{GOpen}(O, m, \sigma)$*, or, in the traceability experiment, output* (I^*, m^*, σ^*)*, such that the identity* id/id^* *output by the opening algorithms of the honest openers in* O/O^* *is so that* $id/id^* \in Q_{\mathsf{GJoin}} \setminus Q_{\mathsf{Corrupt}}$ *(which implies* $id/id^* \neq \perp$*) and* $(id, m, \sigma)/(id^*, m^*, \sigma^*) \notin Q_{\mathsf{GSign}}$*.*

Proof (Sketch). This lemma can be proved by contradiction as follows. On the input of an SDL tuple $(g_1, \tilde{g}_1, g_2 = g^\mu, \tilde{g}_2 = \tilde{g}_1^\mu)$, the idea of the proof is to compute a signature on μsk_{id^*} with $sk_{id^*} \in_R \mathbb{Z}_p^*$ chosen by $\mathcal{B}$ for a random user

identity id^*. If this latter identity is the one for which $\mathcal{A}$ forges a signature, then $\mathcal{B}$ can extract μsk_{id^*}, and thus also μ and win the SDL game.

Since there are at least $t_O + 1$ honest openers in both the anonymity and the traceability games, one can test during issuance whether the adversary broke the soundness of the proofs of correctness openers shares, and abort if it is the case.

To compute a signature on μsk_{id^*}, the reduction algorithm broadcasts $h_{sk} \leftarrow g_2^{r^* sk_{id}}$ instead of $g_1^{r^* sk_{id}}$ where r^* is such that $(a'^*, g_1^{r_*}) \leftarrow \mathcal{H}_0(id^*)$. The fact that at most t_O are corrupt allows to simulate shares of μsk_{id} without the knowledge of μ.

If the adversary can forge a signature on id^* without ever querying her group secret key, one can apply the forking algorithm of the generalized forking lemma [2] to extract μsk_{id^*}. Extracting μsk_{id^*} requires to know the secret key of all dishonest issuers because of the extra dimension added by the PS scalar a'^*. Therefore, one needs to reconstruct them at the setup phase, which is possible given that there are at least $t_I + 1$ honest issuers. Having extracted μsk_{id^*}, the reduction algorithm can compute μ and win the SDL game. □

Theorem 2 (Anonymity). *Scheme* PS-DGS *is anonymous under the first-group DDH and the SDL assumptions over the pairing-group generator* G *for any* $t_O < n_O/2$ *and* $t_I = t_I^* < n_I/2$.

Proof (Sketch). The anonymity of PS-DGS can be proved via the following hybrid argument. Denote by $\mathcal{C}_b$ the challenger of experiment $\mathbf{Exp}^{\mathrm{ano}-b}_{\mathsf{DGS},\lambda,n_I,n_O,t_I,t_O}(\mathcal{A})$. Let Δ be an algorithm that proceeds exactly like $\mathcal{C}_0$ except for the challenge query. To answer the challenge query, Δ sends two random $\mathbb{G}$ elements as re-randomized group elements of the PS signature in $\mathbf{gsk}[id_b^*]$, and programs oracle $\mathcal{H}_1$ to compute a proof of knowledge of sk_{id_b} and a'_b, and complete the challenge group signature.

To show that PS-DGS satisfies anonymity, it suffices to show that $\mathcal{C}_0$ and $\mathcal{C}_1$ are both computationally indistinguishable from Δ under the first-group DDH and the SDL assumptions.

The first-group DDH assumption guarantees that the challenge token is computationally indistinguishable from a uniformly random pair. To prove this, if $(g_1, g_2 = g_1^{\mu}, g_3 = g_1^{\nu}, g_4^{\omega})$ is the tuple received from the DDH challenger, the reduction algorithm guesses two target identities id_0^* and id_1^* for which $\mathcal{A}$ makes the challenge query. For those identities, it compute their secret keys as $\mu sk_{id_0^*}$ and $\mu sk_{id_1^*}$ like in Lemma 2. At the challenge phase, the reduction algorithm computes $\Sigma_2 \leftarrow g_3^{x+y_1 a_0'^*} g_4^{sk_{id_0^*}}$ Note that it requires to reconstruct the PS secret keys shares of the dishonest issuers, and it is possible under the assumption $t_I^* < n_I/2$. If the tuple is a DDH tuple, then it simulates $\mathcal{C}_0$. If it is a random tuple, it simulates Δ.

To guarantee that the answers to GOpen_i queries are indistinguishable from those of the real scheme, one can use random shares during GJoin.U executions for id_0^* and id_1^*. They remain indistinguishable from real shares for the adversary receives only t_O of them.

By Lemma 2, a GOpen_i query can be a forgery on neither id_0^* nor id_1^*. A forgery would be problematic as it would require to simulate opening values without knowing the randomness used to compute the signature being opened. For the other GOpen_i queries, the adversary knows the randomness if the query should open to the guessed target identities. One can then simulate opening values $e(\Sigma_1, \tilde{Y}_0)^{s_i}$ for signature that are not forged, bound to the signature being open via Σ_1 and broadcast during GOpen protocol executions. Indistinguishability is thereby guaranteed. □

Theorem 3 (Traceability). *Denoting by $q_{\mathcal{H}_0}$ the number of $\mathcal{H}_0$ queries, scheme* PS-DGS *satisfies traceability under the $q_{\mathcal{H}_0}$-MSDH-1, the ADH-KE and the SDL assumptions over the pairing-group generator* G *for any $t_I < n_I/2$, $t_O < n_O$ and $t_O^* = \min(t_O, n_O - t_O - 1)$.*

Proof (Sketch). The proof consists in reducing the traceability of PS-DGS to the existential unforgeability of the PS signature scheme. As its unforgeability relies on the $q_{\mathcal{H}_0}$-MSDH-1 assumption, the theorem follows.

In the key-generation phase, the reduction algorithm runs the simulator of the protocol of Gennaro et al. and extract the shares of the corrupt issuers. It later uses these keys to simulate $\mathsf{GJoin.U}$ and $\mathsf{GJoin.I}_i$ queries.

If the adversary wins the traceability game, it outputs a pair (O^*, m^*, σ^*) such that $\mathsf{GVerf}(ipk, m^*, \sigma^*) = 1$ and the signature either opens to an honest identity who never computed it, or the signature fails to open altogether.

By Lemma 2, it cannot open to an honest identity.

Besides, protocol GOpen is unanimous between honest openers. That is, for all (O, m, σ), if id_i denotes the output of the ith manager opening algorithm for $i \in O$, then $id_i = id_j$ for all honest openers with index $i, j \in O$. Indeed, during GOpen protocol executions, the broadcast protocol ensures that all honest managers receive the same opening sets S_j. Unanimity follows from the fact that the opening algorithms test the signature verification equality only on identities for which they all have an entry in their registers, and that they proceed in the same order (i.e., lexicographic order).

Moreover, the identity returned by GOpen must be in Q_{GJoin} for any $\mathsf{GJoin.U}$ or $\mathsf{GJoin.I}_i$ query starts by adding the identity of the query to GJoin.

The only possibility left is that the signature opens to $\perp$. Running the forking algorithm of the generalized forking lemma [2], one can extract a user secret key sk_{id} and an extra scalar a' underlying the valid forgery. As the opening algorithms return $\perp$, at least one honest issuers did not sign it as otherwise the opening algorithms would have a user identity. Signature σ^* therefore a forgery on m^* for the PS signature scheme. □

Discussion. The anonymity of the scheme is only guaranteed if less than half of the issuers are corrupt, and not all as one would hope. One reason is that the generation of the issuer keys is interactive, so for the protocol to terminate, there cannot be a dishonest majority. Another reason is that the reduction to the DDH requires to know all issuer secret keys which are shares of the PS secret

key obtained during the key-generation protocol. To be able to reconstruct the shares of the corrupt issuers, the number $n - t_I^*$ of honest issuers must be greater than t_I^*.

Concerning traceability, it requires the number of corrupt openers not only to be smaller than $n_O - t_O - 1$ as explained in Section 3, but also smaller than t_O. It is due to the fact that even though the openers are separate from the issuers, they must still obtain user secret key shares from the joining protocol to be able to open signatures. In this sense, opening is not completely independent of issuance, and it is precisely what allows the signatures of the scheme to be so short. This constraint on the number of corrupt openers appears in the proof of Lemma 2 in which the forked algorithm must simulate shares of user secret keys, and it can only do so if at most t_O openers are corrupt since the opening threshold is $t_O + 1$.

Efficiency. On a Cocks–Pinch pairing curve [35] defined over a field of order 2^{544} and with embedding degree 8, group elements in $\mathbb{G}$ take 68 Bytes for a group of 256-bit order. Note that this curve provides 131 bits of security [35].

A group signature from our scheme consists of two $\mathbb{G}$ elements and three $\mathbb{Z}_p$ elements, totalling 232 Bytes. The hash value in the proof of knowledge of a multi-signature can actually be shortened to second-preimage resistant length, further shortening a group signature to 216 Bytes.

Considering only group operations, computing a signature costs 4 exponentiations in $\mathbb{G}$ and the product of 2 pairing values. Verifying a signature costs 4 exponentiations in $\mathbb{G}$ and the multiplication of 4 pairing values.

Comparison with other Schemes. Table 1 gives a comparison of our threshold DGS scheme of with other CCA-anonymous dynamic group signatures schemes based on pairings. Lattice-based schemes are absent from the table since have considerably larger signatures than pairing-based ones, and are therefore less preferred for practical applications.

Note that unlike the scheme of Bichsel et al. [8] and the scheme of Pointcheval and Sanders [46, Appendix A.1], our scheme supports threshold issuance and threshold opening, and does not rely on an interactive assumption, but rather a q-type of assumption.

The table features the CL and BBS* scheme. As explained in the introduction, Gennaro et al. extended [31] those schemes to make them support threshold issuance. Since those schemes follow the sign-and-encrypt paradigm, the CL and BBS* scheme also easily allow for threshold opening. Note that generating the decryption key in a distributed and robust way (e.g., with the protocol of Gennaro et al.) would require an honest majority of openers. However, the security guarantees of those scheme in a threshold setting are unclear since Gennaro et al. did not provide a formal model for threshold group signatures schemes.

The table also includes the "Coconut" credential system which is based on the CT-RSA16 version of the PS signature scheme. The first reason is that their system and our scheme are related as presenting a credential consists in proving

Scheme	Sig. Size	Sign	Verify	$t_I <$	$t_O <$
PS-DGS [Section 4]	$2\mathbb{G} + 3\mathbb{Z}_p$	$4\mathbb{G} + 1P^2$	$4\mathbb{G} + 1P^4$	$n_I/2$	$n_O/2$
PS [46]	$2\mathbb{G} + 2\mathbb{Z}_p$	$2\mathbb{G} + 1\mathbb{G}_T$	$3\mathbb{G} + 1P^3$	✗	✗
Bichsel et al. [8]	$3\mathbb{G} + 2\mathbb{Z}_p$	$3\mathbb{G} + 1\mathbb{G}_T$	$1\mathbb{G} + 1\mathbb{G}^2 + 2P^2$	✗	✗
CL [21,31]	$7\mathbb{G} + 4\mathbb{Z}_p$	$11\mathbb{G} + 1\mathbb{G}^2 + 1\mathbb{G}_T^2$	$1\mathbb{G} + 2\mathbb{G}^2 + 2\mathbb{G}^3 + 1\tilde{\mathbb{G}}^2 + 2P^2$	$n_I/2$	$n_O/2$
BBS* [8,31]	$4\mathbb{G} + 5\mathbb{Z}_p$	$5\mathbb{G} + 3\mathbb{G}^2 + 1\mathbb{G}_T^5$	$4\mathbb{G}^2 + 1\mathbb{G}^3 + 1\mathbb{G}^4 + 1P^2$	$n_I/2$	$n_O/2$
Coconut [52]	$3\mathbb{G} + 1\tilde{\mathbb{G}} + 3\mathbb{Z}_p$	$4\mathbb{G} + 1\tilde{\mathbb{G}}^3 + 1\tilde{\mathbb{G}}^2$	$1\mathbb{G} + 1\mathbb{G}^2 + 1\tilde{\mathbb{G}}^4 + P^2$	$n_I/2$	✗

Table 1. Comparison of our PS-DGS scheme with other schemes in terms of signature sizes, and the cost to compute and verify signatures. For the signing and verification costs, $\mathbb{G}^\ell$ indicates an ℓ-exponentiation in $\mathbb{G}$, and similarly for $\tilde{\mathbb{G}}$ and $\mathbb{G}_T$. P denotes the number of pairing computations required, and P^ℓ stands for the product of ℓ pairing values, which is more efficient than computing ℓ pairings separately. The table also gives strict upper bounds for the number of issuers and openers that can be corrupt for traceability and anonymity to hold, respectively. These bounds assume that all key-generation protocols terminate, hence a necessary honest majority. The mention ✗ appears in case a property is not defined (in a distributed setting) for the scheme. Note that the bounds for the CL and BBS* schemes are only purported as there is no formal model for their threshold versions [31].

that a user knows a signature from an authority on her attributes. It is also how our group signatures are computed. It should be noted that the verification of a coconut token involves a multiplication in $\mathbb{G}$, which is approximately as expensive as a squaring, so it was counted as an exponentiation. Their system is further related to ours in that it supports threshold issuance. However, since it is a credential system, there is no need for opening, and it therefore avoids the challenge of also achieving (threshold) opening while maintaining signatures short. Besides, the authors did not provide a security model to analyze their scheme.

One can see that, except for the PS group signatures (of which the non-threshold traceability relies on an interactive assumption) which are not defined in a threshold setting, the signatures of our scheme are the shortest. Indeed, even when compared to the signatures of Bichsel et al. [8], our signatures are shorter since $\mathbb{Z}_p$ elements are typically shorter than $\mathbb{G}$ elements.

5 Distributed Group Signatures from Multi-Signatures

In practice, the ability to open signatures with a threshold number of openers seems to be a natural requirement. For privacy concerns, it is desirable to distribute the role of the opener over many entities. If this number is high, not all potential openers can be expected to be permanently available, or to agree in case of legal dispute for instance. Therefore, only a threshold amount of them should be required to open signatures.

The threshold aspect is, however, less critical for issuance. Although the ability to add users to the group should be *distributed* to several entities for stronger security, the number of issuers generally does not need to be as high as the number of potential openers. Effectively, issuers would be a few service providers, whereas openers would be judges or jurors. Besides, the eventuality of a disagreement about adding a user is less likely than a dispute about opening a signature. Distributed issuance instead of threshold issuance might then be satisfactory for many real-world scenarios.

Distributed Group Signatures. On this account, this section presents a group signature scheme with distributed issuance and threshold opening. The benefit of this dedicated distributed scheme is that the traceability of the scheme now holds even if all issuers but one are corrupt. It means that even if *all but one* issuer collude, they can neither compute an untraceable valid group signature nor forge a group signature that opens to an honest user who never computed it. In contrast, our scheme PS-DGS in Section 4 does not allow to corrupt $n_I - 1$ issuers because of the $t_I < n_I/2$ bound stemming from the key-generation protocol of Gennaro et al. [32].

Restricting issuance to a distributed setting rather than a threshold one allows to dispense with a key-generation protocol. Instead, we base our distributed scheme on a novel multi-signature variant of the PS signature scheme that we first introduce. Relying on multi-signatures not only overcomes the $t_I < n_I/2$ bound but also features a simpler, non-interactive key generation phase.

In terms of signature size and computational costs, this group signature scheme still has the same efficiency as the scheme in Section 4. It also supports the same threshold opening capabilities as the previous schemes.

PS Multi-Signatures. A multi-signature scheme with n signers consists of the following algorithms: $\mathsf{Setup}(1^\lambda, n, aux) \to pp$ a set-up algorithm, $\mathsf{KG}(pp) \to (vk, sk)$ a key-generation algorithm, $\mathsf{KAggreg}(vk_1, \ldots, vk_n) \to avk$ an algorithm which compresses (or aggregates) the verification key of n signers into a single key, $\mathsf{Sign}(sk, m) \to \sigma$ a signing algorithm, $\mathsf{SAggreg}((vk_i)_{i=1}^n, m, (\sigma_i)_{i=1}^n) \to \sigma$ an algorithm which aggregates the signatures of n signers on the same message to form a compact one and $\mathsf{Verf}(avk, m, \sigma) \to b \in \{0, 1\}$ a verification algorithm which verifies an aggregated signature on a message w.r.t. an aggregated verification key. A secure multi-signature scheme ensures that an efficient adversary cannot compute with non-negligible probability a signature on a message and claim that a signer who never signed the message did. See Appendix D for a formalization of the security of multi-signature schemes.

Our distributed group signatures are based on a novel multi-signature variant of the PS signature scheme that constitutes a contribution of independent interest. The scheme supports public-key aggregation and is proved secure in the *plain public-key* model, meaning that signers do not have to prove knowledge of their secret key to prevent rogue key attacks. It is the security of multi-signatures which enables our distributed scheme to withstand the corruption of $n_I - 1$ issuers.

PS multi-signatures further allow for efficient proofs of knowledge like the PS signature scheme, an important feature in the design of privacy-enhancing cryptographic protocols as shown with the following group-signature scheme.

We first present this new multi-signature scheme and then sketch how it can be used in a distributed group-signature scheme.

5.1 Pointcheval–Sanders Multi-Signatures

This section introduces a novel multi-signature scheme based on the CT-RSA'18 version of the Pointcheval–Sanders signature scheme (see Section 2.5). The CT-RSA'18 version of the PS signature scheme is referred to as the *modified* PS scheme, and the unforgeability proof of the multi-signature scheme makes a gradual reduction starting from the *original* PS signatures, i.e., the CT-RSA'16 version [46]. The core difference between the original and modified versions of PS signatures is that the original one can only be proved to be *weakly* unforgeable from the q-MDSH-1 assumption, whereas the modified scheme leverages an extra scalar to lift the security to standard unforgeability.

For each version of their signatures, Pointcheval and Sanders also distinguish the *single-message* variant that allows to sign a single message in $\mathbb{Z}_p$ from the *multi-message* variant that allows to sign blocks of messages. This distinction is also made for PS multi-signatures and is further relevant in the unforgeability proof.

The modified PS signature scheme can be turned into a multi-signature scheme by having each signer generate her pair of keys separately. However, to produce a signature for a given message, the signers need to agree on a common base h and a common extra-message m'. They can do so by hashing the message to be signed with a random oracle $\mathcal{H}_0 \colon \mathbb{Z}_p^k \to \mathbb{Z}_p \times \mathbb{G}^*$ to obtain these[4]. Moreover, to securely aggregate keys, as for BLS multi-signatures [12], another random oracle $\mathcal{H}_1 \colon \tilde{\mathbb{G}}^{n(k+2)} \to \Theta^n \subseteq \mathbb{Z}_p^n$ (with $1/|\Theta|$ negligible in λ) is introduced.

Given a type-3 pairing-group generator G and a security parameter $\lambda \in \mathbb{N}$, the (modified) PS (multi-message) multi-signature scheme in a pairing group $\Gamma = \left(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e\right) \leftarrow \mathsf{G}(1^\lambda)$ consists of the following algorithms:

$\mathsf{PSM.Setup}(1^\lambda, n, k, \Gamma) \to pp$: generate $g \in_R \mathbb{G}^*$, $\tilde{g} \in_R \tilde{\mathbb{G}}^*$ and return $pp \leftarrow (\Gamma, g, \tilde{g}, n, k)$. Integer n is the number of signers, and k is the number of messages to be signed.

$\mathsf{PSM.KG}(pp) \to (vk, sk)$: generate $x, y_1, \ldots, y_{k+1} \in_R \mathbb{Z}_p^*$, compute $\tilde{X} \leftarrow \tilde{g}^x$, $\tilde{Y}_j \leftarrow \tilde{g}^{y_j}$ for $j \in [k+1]$, and set $vk \leftarrow \left(\tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{k+1}\right)$, $sk \leftarrow (x, y_1, \ldots, y_{k+1})$. Return (vk, sk).

[4] Note that Pointcheval and Sanders already suggested [47, Section 4.3] to use a random oracle to generate m' deterministically in their single-signer signature scheme. Using a hash function to generate h is a variation of this idea which has also been considered by Sonnino et al. [52], but with the original PS signature scheme and without proving that the scheme remains secure.

$\mathsf{PSM.KAggreg}\big(vk_1, \ldots, vk_n\big) \to avk$: compute $(t_1, \ldots, t_n) \leftarrow \mathcal{H}_1(vk_1, \ldots, vk_n)$ and return $avk \leftarrow \prod_{i=1}^{n} vk_i^{t_i}$.

$\mathsf{PSM.Sign}\big(sk, \boldsymbol{m} = (m_1, \ldots, m_k)\big) \to \sigma$: compute $(m', h) \leftarrow \mathcal{H}_0(\boldsymbol{m}) \in \mathbb{Z}_p \times \mathbb{G}^*$ and return $\sigma \leftarrow \Big(m', h, h^{x + \sum_{j=1}^{k} y_j m_j + y_{k+1} m'}\Big)$.

$\mathsf{PSM.SAggreg}\big((vk_i)_{i=1}^n, \boldsymbol{m}, (\sigma_i)_{i=1}^n\big) \to \sigma$: parse σ_i as $(m'_i, \sigma_{i,1}, \sigma_{i,2})$ for $i \in [n]$. If $m'_1 = \cdots = m'_n$ and $\sigma_{1,1} = \cdots = \sigma_{n,1}$, compute $(t_1, \ldots, t_n) \leftarrow \mathcal{H}_1(vk_1, \ldots, vk_n)$ and $\sigma_2 \leftarrow \prod_{i=1}^{n} \sigma_{i,2}^{t_i} = \sigma_{1,1}^{\xi + \sum_{j=1}^{k} u_j m_j + u_{k+1} m'}$, with $\xi = \sum_{i=1}^{n} x_i t_i$, $u_j = \sum_{i=1}^{n} y_{i,j} t_i$ for $j \in [k]$ and $u_{k+1} = \sum_{i=1}^{n} y_{i,k+1} t_i$. Return $\sigma \leftarrow \big(m'_1, \sigma_{1,1}, \sigma_2\big)$. Otherwise, return $\perp$.

$\mathsf{PSM.Verf}\big(avk, \boldsymbol{m} = (m_1, \ldots, m_k), \sigma\big) \to b$: parse σ as (m', σ_1, σ_2). If $\sigma_1 \neq 1_{\mathbb{G}_1}$ and $e\left(\sigma_1, \tilde{X} \prod_{i=1}^{k} \tilde{Y}_i^{m_i} \tilde{Y}_{k+1}^{m'}\right) = e(\sigma_2, \tilde{g})$ then return 1, else return 0.

Remark 2. Note that the original PS signature scheme (Appendix E.1) can also be turned into a multi-signature scheme in the random oracle model in the same way, except for m' which is omitted.

Theorem 4. *In the random oracle model, denoting by $q_{\mathcal{H}_0}$ the maximum number of $\mathcal{H}_0$ oracle queries, the PS multi-signature scheme in a pairing group generated by G is EUF-CMA under the $q_{\mathcal{H}_0}$-MSDH-1 assumption over G.*

Proof. Similarly the proof [47, Section 5] of the existential unforgeability under CMA of the modified PS multi-message signature scheme, this proof is in two steps. First, the original single-message multi-signature scheme is proved to be EUF-wCMA secure under the $q_{\mathcal{H}_0}$-MSDH-1 assumption, then the existential unforgeability of the modified multi-message multi-signature scheme under adaptive CMAs is reduced to the weak existential unforgeability of the original single-message multi-signature scheme (notice that each signing query implies an $\mathcal{H}_0$ query).

Lemma 3. *In the random oracle model, denoting by $q_{\mathcal{H}_0}$ the amount of non-adaptive signing queries, the original PS single-message multi-signature scheme is EUF-wCMA secure under the $q_{\mathcal{H}_0}$-MSDH-1 assumption.*

Lemma 4. *If the SDL assumption holds over the group-generator G and the PS single-message multi-signature scheme is EUF-wCMA secure, then the modified PS multi-message multi-signature scheme is EUF-CMA secure.*

These two lemmas are proved in Appendix E.3. Together with the fact that the $q_{\mathcal{H}_0}$-MSDH-1 assumption implies the SDL assumption, they imply the theorem. □

Proving Knowledge of a PS Multi-Signature. Proving knowledge of a PS multi-signature valid w.r.t. an aggregated verification key *avk* can be done exactly as in Section 2.5 for proving knowledge of a PS signature valid w.r.t. a verification key *vk*.

5.2 Our Distributed Group Signatures with Threshold Opening

Building on PS multi-signatures, we can now describe our dynamic group signatures with distributed issuance and threshold opening. The main differences compared to the scheme in Section 4 are in the key-generation phase and the issuance protocol.

Key Generation. Given that PS multi-signatures support public key aggregation, the issuers can now generate their PS keys separately instead of executing a protocol. The PS public keys generated by the issuers can later be aggregated with a random oracle $\mathcal{H}_1$ to obtain a global issuer public key that can be used for signing and verifying signatures.
Nevertheless, the security proofs still require to be able to extract the keys of dishonest issuers, so they additionally have to prove knowledge of their PS secret keys. It is worth stressing that proving knowledge of secret keys is not needed for the stand-alone PS multi-signature scheme, but rather needed to prove the group-signature scheme secure.
The opener keys are generated as in the scheme in Section 4.
The group public key is now the concatenation of all issuer-and-opener public keys.

Issuance. During issuance, each issuer blindly signs h_{sk} with his secret key. After receiving signatures from all issuers, the user aggregates them and verifies their validity with respect to the global issuer public key, i.e., the PS aggregated public key. Therefore, user certificates are now PS multi-signatures instead of PS signatures.

Update Registers, Sign, Verify & Open. Updating registers, and computing, verifying and opening signatures are done as in the scheme of Section 4, except that signing and verifying are now done with another hash function $\mathcal{H}_2$ modeled as a random oracle.

We formally describe the key-generation and key-aggregation algorithms, and the issuance protocol in Appendix F.

Security. The anonymity of the scheme holds under the DDH and the SDL assumptions over the group generator if $t_O < n_O/2$. It is not necessary to assume that less than half of the issuers are corrupt since the issuer secret keys can be extracted from their proofs of knowledge.

As for traceability, it holds under the $q_{\mathcal{H}_0}$-MSDH-1, the ADH-KE and the SDL assumptions if at most $n_I - 1$ issuers are corrupt. It is because no colluding $n_I - 1$ PS signers can, with non-negligible probability, compute a multi-signature that is valid w.r.t. the aggregated key of all the n_I signers. The assumption that at most $\min(t_O, n_O - t_O - 1)$ openers are corrupt is still necessary though. Indeed, no more than t_O openers can be corrupt in the simulation of shares of μsk_{id} in the proof that no honest user can be framed by $n_I - 1$ corrupt issuers; and the winning condition still depends on a correct execution of protocol GOpen which requires the non-corrupt registers of at least $t_O + 1$ openers.

Acknowledgements. The authors thank David Pointcheval for helpul discussions. This work supported by the CHIST-ERA USEIT project and the EU H2020 Research and Innovation Program under Grant Agreement No. 786725 (OLYMPUS).

References

1. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 255–270. Springer, Heidelberg, Aug. 2000.
2. A. Bagherzandi, J. H. Cheon, and S. Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In P. Ning, P. F. Syverson, and S. Jha, editors, *ACM CCS 08*, pages 449–458. ACM Press, Oct. 2008.
3. M. Bellare, G. Fuchsbauer, and A. Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, Dec. 2016.
4. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003.
5. M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06*, pages 390–399. ACM Press, Oct. / Nov. 2006.
6. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993.
7. M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In A. Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 136–153. Springer, Heidelberg, Feb. 2005.
8. P. Bichsel, J. Camenisch, G. Neven, N. P. Smart, and B. Warinschi. Get shorty via group signatures without encryption. In J. A. Garay and R. D. Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 381–398. Springer, Heidelberg, Sept. 2010.
9. J. Blömer, J. Juhnke, and N. Löken. Short group signatures with distributed traceability. In I. S. Kotsireas, S. M. Rump, and C. K. Yap, editors, *Mathematical Aspects of Computer and Information Sciences*, pages 166–180, Cham, 2016. Springer International Publishing.
10. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Heidelberg, Jan. 2003.
11. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, Aug. 2004.
12. D. Boneh, M. Drijvers, and G. Neven. Compact multi-signatures for smaller blockchains. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 435–464. Springer, Heidelberg, Dec. 2018.
13. D. Boneh, S. Eskandarian, and B. Fisch. Post-quantum EPID group signatures from symmetric primitives. Cryptology ePrint Archive, Report 2018/261, 2018. `https://eprint.iacr.org/2018/261`.

14. D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In V. Atluri, B. Pfitzmann, and P. McDaniel, editors, *ACM CCS 04*, pages 168–177. ACM Press, Oct. 2004.
15. D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. 2017.
16. C. Boschini, J. Camenisch, and G. Neven. Floppy-sized group signatures from lattices. In B. Preneel and F. Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 163–182. Springer, Heidelberg, July 2018.
17. E. F. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In V. Atluri, B. Pfitzmann, and P. McDaniel, editors, *ACM CCS 04*, pages 132–145. ACM Press, Oct. 2004.
18. J. Camenisch, L. Chen, M. Drijvers, A. Lehmann, D. Novick, and R. Urian. One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. In *2017 IEEE Symposium on Security and Privacy*, pages 901–920. IEEE Computer Society Press, May 2017.
19. J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In C. Blundo and S. Cimato, editors, *SCN 04*, volume 3352 of *LNCS*, pages 120–133. Springer, Heidelberg, Sept. 2005.
20. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Heidelberg, Aug. 2002.
21. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, Aug. 2004.
22. M. Chase and A. Lysyanskaya. On signatures of knowledge. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Heidelberg, Aug. 2006.
23. D. Chaum and E. van Heyst. Group signatures. In D. W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, Heidelberg, Apr. 1991.
24. D. Derler and D. Slamanig. Highly-efficient fully-anonymous dynamic group signatures. In J. Kim, G.-J. Ahn, S. Kim, Y. Kim, J. López, and T. Kim, editors, *ASIACCS 18*, pages 551–565. ACM Press, Apr. 2018.
25. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, Aug. 1990.
26. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th FOCS*, pages 427–437. IEEE Computer Society Press, Oct. 1987.
27. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, Aug. 1987.
28. I. O. for Standardization, 2015.
29. G. Fuchsbauer and M. Orrù. Non-interactive zaps of knowledge. In B. Preneel and F. Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 44–62. Springer, Heidelberg, July 2018.
30. T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03. ACM.
31. R. Gennaro, S. Goldfeder, and B. Ithurburn. Fully distributed group signatures, 2019.

32. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 295–310. Springer, Heidelberg, May 1999.
33. E. Ghadafi. Efficient distributed tag-based encryption and its application to group signatures with efficient distributed traceability. In D. F. Aranha and A. Menezes, editors, *LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 327–347. Springer, Heidelberg, Sept. 2015.
34. T. C. Group. Trusted platform module library specification,family "2.0", 2014.
35. A. Guillevic, S. Masson, and E. Thomé. Cocks–pinch curves of embedding degrees five to eight and optimal ate pairing computation. 2019.
36. B. Libert, S. Ling, F. Mouhartem, K. Nguyen, and H. Wang. Adaptive oblivious transfer with access control from lattice assumptions. In T. Takagi and T. Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 533–563. Springer, Heidelberg, Dec. 2017.
37. B. Libert and M. Yung. Dynamic fully forward-secure group signatures. In D. Feng, D. A. Basin, and P. Liu, editors, *ASIACCS 10*, pages 70–81. ACM Press, Apr. 2010.
38. S. Ling, K. Nguyen, H. Wang, and Y. Xu. Lattice-based group signatures: Achieving full dynamicity with ease. In D. Gollmann, A. Miyaji, and H. Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 293–312. Springer, Heidelberg, July 2017.
39. S. Ling, K. Nguyen, H. Wang, and Y. Xu. Constant-size group signatures from lattices. In M. Abdalla and R. Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 58–88. Springer, Heidelberg, Mar. 2018.
40. M. Manulis. Democratic group signatures on example of joint ventures. Cryptology ePrint Archive, Report 2005/446, 2005. `http://eprint.iacr.org/2005/446`.
41. M. Manulis, N. Fleischhacker, F. Günther, F. Kiefer, and B. Poettering. Group signatures: Authentication with privacy, 2012.
42. G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, 2019.
43. S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures: Extended abstract. In *ACM CCS 01*, pages 245–254. ACM Press, Nov. 2001.
44. G. Neven, G. Baldini, J. Camenisch, and R. Neisse. Privacy-preserving attribute-based credentials in cooperative intelligent transport systems. In *2017 IEEE Vehicular Networking Conference, VNC 2017*, pages 131–138. IEEE, 2017.
45. J. Petit, F. Schaub, M. Feiri, and F. Kargl. Pseudonym schemes in vehicular networks: A survey. *IEEE Communications Surveys and Tutorials*, 17(1):228–255, 2015.
46. D. Pointcheval and O. Sanders. Short randomizable signatures. In K. Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Heidelberg, Feb. / Mar. 2016.
47. D. Pointcheval and O. Sanders. Reassessing security of randomizable signatures. In N. P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 319–338. Springer, Heidelberg, Apr. 2018.
48. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.
49. C.-P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, Aug. 1990.
50. C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, Jan. 1991.
51. A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, Nov. 1979.

52. A. Sonnino, M. Al-Bassam, S. Bano, and G. Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. *CoRR*, abs/1802.07344, 2018.
53. W. Whyte, A. Weimerskirch, V. Kumar, and T. Hehn. A security credential management system for V2V communications. In *2013 IEEE Vehicular Networking Conference*, pages 1–8. IEEE, 2013.
54. D. Zheng, X. Li, C. Ma, K. Chen, and J. Li. Democratic group signatures with threshold traceability. Cryptology ePrint Archive, Report 2008/112, 2008. `http://eprint.iacr.org/2008/112`.

A Generalized Forking Lemma

This section presents the generalized forking lemma which provides key arguments in the security proofs of our group signatures.

The original forking lemma was formulated by Pointcheval and Stern [48] to analyze the security of Schnorr signatures [50]. The lemma rewinds a forger $\mathcal{A}$ against the Schnorr signature scheme in the random-oracle model (ROM) to a "crucial" random-oracle query (typically, the query involved in a forgery) and runs $\mathcal{A}$ again from the crucial query with fresh random-oracle responses. The lemma essentially translates that if $\mathcal{A}$ has non-negligible success probability in a single run, then the forking algorithm will generate two successful executions with non-negligible probability.

Bellare and Neven [5] generalized the forking lemma to apply to any algorithm $\mathcal{A}$ in the random-oracle model using a single rewinding, while Bagherzandi, Cheon, and Jarecki [2] generalized the lemma even further to multiple subsequent rewindings on multiple crucial queries. This section recalls a slight modification of the latter version.

Let $\mathcal{A}$ be an algorithm that is given an input in as well as randomness $f = (\rho, h_1, \ldots, h_{q_\mathrm{H}})$, where ρ is $\mathcal{A}$'s random tape and $h_1, \ldots, h_{q_\mathrm{H}}$ are random values from $\mathbb{Z}_q$. Let Ω be the space of all such vectors f and let $f|_i = (\rho, h_1, \ldots, h_{i-1})$. Consider an execution of $\mathcal{A}$ on input in and randomness f with access to oracle $\mathcal{O}$, denoted $\mathcal{A}^{\mathcal{O}}(in, f)$, as *successful* if it outputs a tuple $(J, \{out_j\}_{j\in J}, aux)$, where J is a multi-set that is a non-empty subset of $\{1, \ldots, q_\mathrm{H}\}$, $\{out_j\}_{j\in J}$ is a multi-set of side outputs, and aux is an additional set of auxiliary outputs. $\mathcal{A}$ is said to have failed if it outputs $J = \emptyset$. Let ε be the probability that $\mathcal{A}(in, f)$ is successful for fresh randomness $f \leftarrow_\$ \Omega$ and for an input $in \leftarrow_\$ \mathsf{IG}$ generated by an input generator IG.

For a given input in, the generalized forking algorithm $\mathcal{GF}_\mathcal{A}$ is defined as follows:

$\mathcal{GF}_\mathcal{A}(in)$:
$f = (\rho, h_1, \ldots, h_{q_\mathrm{H}}) \leftarrow_\$ \Omega$
$(J, \{out_j\}_{j\in J}, aux) \leftarrow \mathcal{A}^{\mathcal{O}}(in, f)$
If $J = \emptyset$ then return fail
$Aux \leftarrow aux$
Let $J = \{j_1, \ldots, j_n\}$ such that $j_1 \leq \ldots \leq j_n$

For $i = 1, \ldots, n$ do
$succ_i \leftarrow 0$; $k_i \leftarrow 0$; $k_{\max} \leftarrow 8nq_{\mathrm{H}}/\varepsilon \cdot \ln(8n/\varepsilon)$
Repeat until $succ_i = 1$ or $k_i > k_{\max}$
$f'' \leftarrow_{\$} \Omega$ such that $f''|_{j_i} = f|_{j_i}$
Let $f'' = (\rho, h_1, \ldots, h_{j_i - 1}, h''_{j_i}, \ldots, h''_{q_{\mathrm{H}}})$
$(J'', \{out''_j\}_{j \in J''}, aux) \leftarrow \mathcal{A}^{\mathcal{O}}(in, f'')$
$Aux \leftarrow Aux \cup aux$
If $h''_{j_i} \neq h_{j_i}$ and $J'' \neq \emptyset$ and $j_i \in J''$ then
$out'_{j_i} \leftarrow out''_{j_i}$; $succ_i \leftarrow 1$
If $succ_i = 1$ for all $i = 1, \ldots, n$
Then return $(J, \{out_j\}_{j \in J}, \{out'_j\}_{j \in J}, Aux)$
Else return fail

$\mathcal{GF}_{\mathcal{A}}$ is considered successful if it does not return fail. The main difference to Bagherzandi et al.'s forking lemma [2] is $\mathcal{A}$'s access to the oracle $\mathcal{O}$ and the additional auxiliary output aux that gets accumulated in Aux over all runs of $\mathcal{A}$, including failed runs. If the oracle $\mathcal{O}$ is deterministic, meaning that it always answers the same query with the same response, it is easy to see that these extensions do not impact the bounds of their forking lemma, so the following statement still holds.

Lemma 5 (Generalized Forking Lemma [2]). *Let* IG *be a randomized algorithm and* $\mathcal{A}$ *be a randomized algorithm running in time* τ *with access to a deterministic oracle* $\mathcal{O}$ *that succeeds with probability* ε*. If* $q > 8nq_{\mathrm{H}}/\varepsilon$*, then* $\mathcal{GF}_{\mathcal{A}}(in)$ *runs in time at most* $\tau \cdot 8n^2 q_{\mathrm{H}}/\varepsilon \cdot \ln(8n/\varepsilon)$ *and succeeds with probability at least* $\varepsilon/8$*, where the probability is over the choice of in* $\leftarrow_{\$}$ IG *and over the coins of* $\mathcal{GF}_{\mathcal{A}}$*.*

B Security of our Threshold Dynamic Group Signatures

In this section, we prove the lemmas and theorems in Section 4.

Proof (of Lemma 1 – Unanimity). During GOpen protocol executions, the broadcast protocol and the fact that the ledger is publicly available ensure that all honest openers receive the same opening sets S_j. Moreover, the opening algorithms test the signature verification equality only on identities for which they all have an entry in their registers. As they proceed in the same order (i.e., lexicographic order), the claim follows. □

Proof (of Lemma 2 – Forgery). This lemma can be proved by contradiction as follows. Assume that there exists an adversary $\mathcal{A}$ that computes such a forgery in either of the games with probability at least ε and in time $\tau_{\mathcal{A}}$. Assume $p > 8q_{\mathcal{H}_1}/\varepsilon$. Consider a reduction algorithm $\mathcal{B}$ which interacts with the SDL challenger.

On the input of an SDL tuple $(g, \tilde{g}, g^{\mu}, \tilde{g}^{\mu})$, the idea of the proof is to compute a PS signature on μsk_{id^*} with $sk_{id^*} \in_R \mathbb{Z}_p^*$ chosen by $\mathcal{B}$ for a random user identity

id^*. If this identity is the one for which $\mathcal{A}$ forges a signature, then $\mathcal{B}$ can extract μsk_{id^*}, so also μ and win the SDL game. To extract μsk_{id^*}, algorithm $\mathcal{B}$ runs the forking algorithm of the generalized forking lemma (Lemma 5) on an algorithm $\mathcal{A}'$ of which $\mathcal{A}$ is a sub-routine.

In more detail, $\mathcal{A}'$ runs on the input of public parameters pp as in the scheme, of a tuple $(g_1, \tilde{g}_1, g_2, \tilde{g}_2) \in \left(\mathbb{G}^* \times \tilde{\mathbb{G}}^*\right)^2$ and of a random tape ρ.

$\mathcal{A}'$ answers random-oracle queries as follows:

- $\mathcal{H}_0(id)$: forward the query to $\mathcal{B}$.
- $\mathcal{H}_1(b \in \{0,1\}^*)$: if $(b, *) \notin Q_{\mathcal{H}_1}$ then generate $c \in_R \mathbb{Z}_p$, do $ctr \leftarrow ctr + 1$, add (b, c, ctr) to $Q_{\mathcal{H}_1}$ and return c; else retrieve (b, c, ι) from $Q_{\mathcal{H}_1}$ and return c.

During the key-generation phase, $\mathcal{A}'$ proceeds as in the real scheme. At the end of the protocol, it reconstructs the secret keys x, y_0 and y_1 using the shares of the honest issuers. It can do so since there are at least $t_I + 1$ honest issuers by assumption. If $y_0 = 0 \mod p$, which occurs with probability at most $1/p$, algorithm $\mathcal{B}$ aborts its interaction with $\mathcal{A}$ and sends a uniformly random $\mathbb{Z}_p^*$ element to the SDL challenger.

$\mathcal{A}'$ then answers game-oracle queries as follows (if a check fails or if it ever aborts, $\mathcal{A}'$ returns $(\emptyset, \emptyset, \emptyset)$):

- $\mathsf{GJoin.U}(id, I)$: check that $I \in \binom{[n_I]}{t_I+1}$. Add id to Q_{GJoin}. A GJoin execution is triggered and $\mathcal{B}$ plays the role of the user and of the honest issuers.
  - if $id \neq id^*$, follow the protocol but also save sk_{id} in $\mathbf{reg}_j[id]$ for each honest opener $\mathcal{O}_j$.
  - if $id = id^*$, first check that $id^* \notin st_i$ for all honest issuer $\mathcal{I}_i$. Make an internal query $(a'^*, r^*) \leftarrow \mathcal{H}_0(id^*)$. Next, generate $sk_{id^*} \in_R \mathbb{Z}_p^*$ and shares $(s_i)_{i \in [n_O]}$ of $0_{\mathbb{Z}_p}$ (, so $\sum_{j \in I} s_j w_j = 0 \mod p$ for all $O \in \binom{[n_O]}{t_O+1}$). Compute $h_{sk} \leftarrow g_2^{r^* sk_{id^*}} = g_1^{r^* \mu sk_{id^*}}$ and $g_{sk} \leftarrow g_2^{sk_{id^*}}$. To compute verification values $(h_1, \ldots, h_{t_O})$ that convince all corrupt issuers and openers that s_i is a valid share of $\mathrm{dlog}_{g_1^{r^*}}(h_{sk}) = \mu sk_{id^*}$, solve via Gaussian elimination the linear system (with the notation $x \cdot g \coloneqq g^x$ for $x \in \mathbb{Z}_p$ and $g \in \mathbb{G}$)

$$\left[i \cdots i^{t_O}\right]_{i \in \mathcal{CO}} \begin{bmatrix} h_1 \\ \vdots \\ h_{t_O} \end{bmatrix} = \left[(g_1^{r^*})^{s_i} (h_{sk}^*)^{-1}\right]_{i \in \mathcal{CO}}$$

    with unknowns $h_1, \ldots, h_{t_O}$ and $\mathcal{CO}$ as the set of corrupt openers. This linear system has a solution since $\left[i \cdots i^{t_O}\right]_{i \in \mathcal{CO}}$ is full-rank, for it is a sub-matrix of a Vandermonde matrix and the positions $i \in \mathcal{CO}$ are pairwise distinct. It should be noted that unlike the case of the real protocol, there is no relation between the value μsk_{id}^* signed by the issuers and the shares s_i.

Now, program $\mathcal{H}_1$ to simulate a proof of knowledge π of $\mathrm{dlog}_{g_1}(g_{sk}) = \mathrm{dlog}_{g_1^{r^*}}(h_{sk})$. For $i \in \mathcal{CO}$, compute as in the scheme ciphertexts $\tilde{C}_i$ using the $h_1, \ldots, h_{t_O}$ values obtained by solving the linear system, as well as proofs π_i. For each honest opener i, choose uniformly random $\tilde{C}_i$ and program oracle $\mathcal{H}_1$ to simulate proofs of knowledge π_i.
Set $L[id]$ as in the real protocol, and playing the role of the user, broadcast $\mathsf{written}$. Under the DDH assumption over $\tilde{\mathbb{G}}$, from the point of view of $\mathcal{A}$, the distribution of $L[id]$ is indistinguishable from its distribution in the real protocol since $\mathcal{A}$ can decrypt at most t_O shares. The distinguishing advantage of $\mathcal{A}$ is at most n_O times the advantage of $\mathcal{B}$ running it as a sub-routine and attempting to win DDH game in $\tilde{\mathbb{G}}$.
Compute signature shares on μsk_{id^*} (using h_{sk}) for each the honest issuers and send them to the user after adding id^* to their states. Upon receiving signature shares from $\mathcal{A}$, reconstruct the PS signature and verify the result as in the proof of each of the properties (anonymity of traceability). The resulting signature is

$$\left(a'^*, \Sigma_1^* \leftarrow g_1^{r^*}, \Sigma_2^* \leftarrow \left(g_1^{r^*}\right)^{x+y_0 \mu sk_{id^*}+y_1 a'^*}\right),$$

i.e., it is a signature on μsk_{id^*}. Set $\mathbf{gsk}[id^*] \leftarrow (\perp, (a'^*, \Sigma_1^*, \Sigma_2^*))$ and for all honest $\mathcal{O}_j$, set $\mathbf{reg}_j[id^*]$ as in the scheme but also append $(s_i)_{i=1}^{n_O}$.

- $\mathsf{GJoin.I}_i(id, I)$: check that $i \in I \in \binom{[n_I]}{t_I+1}$. Add id to Q_{GJoin} and Q_{Corrupt} (note that $id \neq id^*$ in the event in which the adversary outputs a forgery, necessarily). A GJoin protocol execution is triggered. Play the role of the honest issuers and follow the protocol. If the protocol succeeds, reconstruct $\tilde{Y}_0^{sk_{id}}$ with the shares given by the adversary (recall that there are at least $t_O + 1$ honest openers in either game) for issuers in I, and append $\tilde{Y}_0^{sk_{id}}$ to $\mathbf{reg}_j[id]$ for each honest opener $\mathcal{O}_j$. If the reconstructed value is not consistent with the share of an honest opener (which can be verified with the pairing), abort the protocol for it means that $\mathcal{A}$ broke the soundness of the zero-knowledge proof. It occurs with probability at most $(q_{\mathcal{H}_1}+1)/p$. The probability that it occurs for any of the identities is then at most $|ID|(q_{\mathcal{H}_1} + 1)/p$.
- $\mathsf{GSign}(id, m)$: check that $id \in Q_{\mathsf{GJoin}} \setminus Q_{\mathsf{Corrupt}}$. If $id \neq id^*$ then compute $\sigma \leftarrow \mathsf{GSign}(ipk, \mathbf{gsk}[id], m)$, add (id, m, σ) to Q_{GSign} and return σ. If $id = id^*$ then fetch $(\perp, (a'^*, \Sigma_1^*, \Sigma_2^*)) \leftarrow \mathbf{gsk}[id^*]$, generate $\alpha \in_R \mathbb{Z}_p^*$, compute $(\Sigma_1', \Sigma_2') \leftarrow (\Sigma_1^*, \Sigma_2^*)^\alpha$, simulate a proof π of knowledge of μsk_{id}^* and a'^* by programming $\mathcal{H}_1$, add $(id, m, \sigma = (\Sigma_1', \Sigma_2', \pi))$ to Q_{GSign}, and return σ.
- $\mathsf{GOpen}_i(O, m, \sigma = (\Sigma_1, \Sigma_2, \pi))$: check that $i \in O \in \binom{[n_O]}{t_O+1}$. Add (m, σ) to Q_{GOpen}. A GOpen execution is triggered. Play the role of the honest openers in O. Check that $\mathsf{GVerf}(ipk, m, \sigma) = 1$. Update the registers as in the real scheme. Test the equality

$$e\left(\Sigma_1, \tilde{g}_1^{x+y_1 a'^*} \tilde{g}_2^{y_0 sk_{id^*}}\right) = e(\Sigma_2, \tilde{g}_1).$$

* If the equality does not hold but $e\left(\Sigma_1, \tilde{g}_1^{x+y_1 a'}\, \tilde{Y}_0^{sk_{id}}\right) = e(\Sigma_2, \tilde{g}_1)$ for an identity $id \neq id^* \in Q_{\mathsf{GJoin}} \setminus Q_{\mathsf{Corrupt}}$, where $(a', h) \leftarrow \mathcal{H}_0(id)$, then $\mathcal{A}$ forged a signature on an honest identity $id \neq id^*$ for which it does not have the secret key. Abort the interaction with $\mathcal{A}$ and send a uniformly random $\mathbb{Z}_p^*$ element to the SDL challenger. This is to make sure that the first identity for which $\mathcal{A}$ forges a signature is id^* with probability $1/|ID|$.
* If it holds, if $id^* \in Q_{\mathsf{GJoin}}$ and if $(id^*, m, \sigma) \notin Q_{\mathsf{GSign}}$ then $\mathcal{A}$ has forged σ on m for id^*.
  Parsing σ as $(\Sigma_1, \Sigma_2, \pi = (c, v_{sk}, v_{a'}))$, let ι_σ be the computation step at which $\mathcal{A}$ queried $\mathcal{H}_1$ on $b_\sigma \leftarrow (ipk, \Sigma_1, \Sigma_2, u, m)$ with

$$u \coloneqq e\left(\Sigma_1^{v_{sk}}, \tilde{Y}_0\right) e\left(\Sigma_1^{v_{a'}}, \tilde{Y}_1\right) e\left(\Sigma_1^{-c}, \tilde{X}\right) e\left(\Sigma_2^{c}, \tilde{g}_1\right),$$

  be it the step at which it outputs the forgery, i.e., ι_σ is such that $(b_\sigma, c, \iota_\sigma) \in Q_{\mathcal{H}_1}$. Algorithm $\mathcal{A}'$ sets $J \leftarrow \{\iota_\sigma\}$ and $aux \leftarrow \emptyset$, and returns $(J, \{\sigma\}, aux)$.
* If the equality holds and if $(id^*, m, \sigma) \in Q_{\mathsf{GSign}}$ (which implies $id^* \in Q_{\mathsf{GJoin}}$) then fetch $\mathbf{reg}_i[id^*] = (*, (s_i)_{i=1}^{n_O})$. For $j \in I \setminus \{i\}$, compute $T_{id^*,j} \leftarrow e\left(\Sigma_1, \tilde{Y}_0\right)^{s_j}$. Compute

$$T_{id^*,i} \leftarrow \left(e\left(\Sigma_1, \tilde{g}_2^{y_0 sk_{id^*}}\right) \prod_{j \in I \setminus \{i\}} T_{id^*,j}^{-w_j}\right)^{1/w_i}$$
$$= e\left(\Sigma_1, \tilde{Y}_0\right)^{\left(\mu sk_{id^*} - \sum_{j \neq i} s_j w_j\right)/w_i} = e\left(\Sigma_1, \tilde{Y}_0\right)^{\mu sk_{id^*}/w_i + s_i}.$$

  The values $(T_{id^*,j})_{j \in I}$ are indistinguishable from the values in the real protocol since $\mu sk_{id^*}/w_i + s_i$ and $(s_j)_{j \in I \setminus \{i\}}$ are valid shares of μsk_{id^*} (recall that $\sum_{j \in I} s_j w_j = 0$ for all $O \in \binom{[n_O]}{t_O+1}$). Therefore, with $T_{id^*,i}$ thus computed, the corrupt participating openers, if they follow the protocol, will open to id^*. For the identities $id \neq id^*$ such that $\mathbf{reg}_i[id] \neq \perp$, compute the $T_{id,i}$ values as in the scheme. Follow the rest of the protocol.
* If the equality does not hold and $id^* \in Q_{\mathsf{GJoin}}$, proceed as in the previous case in which the equality held and $(id^*, m, *) \in Q_{\mathsf{GSign}}$. The same indistinguishability arguments apply.
* If the equality does not hold and $id^* \notin Q_{\mathsf{GJoin}}$ then follow the protocol.

– $\mathsf{RevealIU}(id)$: if $id = id^*$ then abort the interaction with $\mathcal{A}$ and send a uniformly random $\mathbb{Z}_p^*$ element to the SDL challenger. Add id to Q_{RevealIU}. Return $\mathbf{gsk}[id]$.
– $\mathsf{ReadReg}(i, id)$: (in the traceability experiment only) return $\mathbf{reg}_i[id]$.
– $\mathsf{WriteReg}(i, id, v)$: (in the anonymity experiment only) set $\mathbf{reg}_i[id] \leftarrow v$.
– In the anonymity experiment $\mathbf{Exp}^{\mathrm{ano}-b}_{\mathsf{DGS},\lambda,n_I,n_O,t_I,t_O}(\mathcal{A})$ for $b \in \{0,1\}$, at the challenge phase, if $id^* \neq id_b$ then return $\mathsf{GSign}(ipk, \mathbf{gsk}[id_b], m^*)$. If

$id_b = id^*$ then generate $\alpha \in_R \mathbb{Z}_p^*$, compute $\Sigma_1 \leftarrow g_1^{\alpha}$ and $\Sigma_2 \leftarrow \Sigma_1^{x+y_1 a'^*} (g_2)^{\alpha y_2 sk^*_{id}}$. Simulate a proof π of knowledge of μsk_{id^*} and a'^* by programming $\mathcal{H}_1$. Return $(\Sigma_1, \Sigma_2, \pi)$.

If $\mathcal{A}$ never forges a signature for an honest identity, $\mathcal{A}'$ returns $(\emptyset, \emptyset, \emptyset)$.

The reduction algorithm $\mathcal{B}$ then proceeds as follows. Upon receiving Γ and $(g_1, \tilde{g}_1, g_2, \tilde{g}_2)$ from the challenger, $\mathcal{B}$ sets $pp_{\mathsf{PS}} \leftarrow (\Gamma, \tilde{g}_1, 1)$ and $pp \leftarrow (pp_{\mathsf{PS}}, g, n_I, n_O, t_I, t_O)$. It then runs $\mathcal{A}'$ on the input of pp, $(g_1, \tilde{g}_1, g_2, \tilde{g}_2)$ and a uniformly random tape ρ.

Throughout its interaction with $\mathcal{A}'$, algorithm $\mathcal{B}$ answers random-oracle queries as follows:

- $\mathcal{H}_0(id)$: if $(id, *) \notin Q_{\mathcal{H}_0}$ then forward the query to $\mathcal{C}$, receive (a', h), add $(id, (a', h))$ to $Q_{\mathcal{H}_0}$ and return (a', h); else retrieve $(id, (a', h))$ from $Q_{\mathcal{H}_0}$ and return (a', h).

Recall that $p > 8tq_{\mathcal{H}_1}/\varepsilon$ by assumption. With probability at least $\varepsilon/8$ and with running time at most $8q_{\mathcal{H}_1}/\varepsilon \cdot \ln(8/\varepsilon) \cdot \tau_{\mathcal{A}}$, the forking algorithm of the generalized forking lemma (Lemma 5) applied to $\mathcal{A}'$ returns $(\{\iota_\sigma\}, \{\sigma\}, \{\sigma'\}, Aux)$. That is, it returns two forgeries for honest identities.

Parsing σ as $(\Sigma_1, \Sigma_2, \pi = (c, v_{sk}, v_{a'}))$, the computation step at which $\mathcal{A}$ queried $\mathcal{H}_1$ on $(ipk, \Sigma_1, \Sigma_2, u, m)$ with

$$u \coloneqq e\left(\Sigma_1^{v_{sk}}, \tilde{Y}_0\right) e\left(\Sigma_1^{v_{a'}}, \tilde{Y}_1\right) e\left(\Sigma_1^{-c}, \tilde{X}\right) e\left(\Sigma_2^{c}, \tilde{g}_1\right)$$

is the same step at which it queried $\mathcal{H}_1$ on $(gpk, \Sigma_1', \Sigma_2', u', m')$ for u' similarly defined. The answers c and c' to these queries are also distinct modulo p.

As the inputs to $\mathcal{A}$ and its randomness are identical until that $\mathcal{H}_1$ query, $\Sigma_1 = \Sigma_1'$, $\Sigma_2 = \Sigma_2'$, $u = u'$ and $m = m'$ necessarily. It also implies $id^* = id'$ by the initial equality test. It follows that

$$\begin{aligned} e\left(\Sigma_1, \tilde{Y}_0^{(v_{sk} - v'_{sk})/(c'-c)} \tilde{Y}_1^{(v_{a'} - v'_{a'})/(c'-c)}\right) &= e(\Sigma_2, \tilde{g}_1) e\left(\Sigma_1, \tilde{X}\right)^{-1} \\ &= e\left(\Sigma_1, \tilde{g}_1^{y_1 a'^*} \tilde{g}_2^{y_0 sk_{id^*}}\right) \end{aligned}$$

with the second equality due to the initial test equality. Therefore,

$$\tilde{g}_1^{\left(y_0(v_{sk} - v'_{sk}) + y_1(v_{a'} - v'_{a'})\right)/(c'-c)} = \tilde{g}_1^{y_1 a'^*} \tilde{g}_2^{y_0 sk_{id^*}}.$$

Compute and send to the SDL challenger the value

$$\left(y_0(v_{sk} - v'_{sk}) + y_1\left((v_{a'} - v'_{a'}) - a'^*(c' - c)\right)\right)/(c' - c) y_0 sk_{id^*},$$

thereby winning the SDL game.

In the event in which $\mathcal{A}$ forges with non-negligible probability a signature that opens to an honest user for which it has not queried the secret keys, the

first identity for which it does is id^* with probability $1/|ID|$. Algorithm $\mathcal{B}$ then wins the SDL game with probability at least

$$1/|ID| \cdot \varepsilon/8 \cdot (1 - 1/p) - |ID|(q_{\mathcal{H}_1} + 1)/p - n_O|ID|\mathbf{Adv}^{\mathrm{DDH}-2}_{\mathsf{G},\lambda}(\mathcal{B}(\mathcal{A}))$$

and with running time at most $8q_{\mathcal{H}_1}/\varepsilon \cdot \ln(8/\varepsilon) \cdot \tau_{\mathcal{A}} + O(|ID|)$. If ε were negligible, $\mathcal{B}$ would contradict the SDL assumption. Consequently, ε is necessarily negligible. □

Proof (of Theorem 2 – Anonymity). The anonymity of DGS can be proved via the following hybrid argument. Denote by $\mathcal{C}_b$ the challenger of experiment $\mathbf{Exp}^{\mathrm{ano}-b}_{\mathsf{DGS},\lambda,n_I,n_O,t_I,t_O}(\mathcal{A})$. Let Δ be an algorithm that proceeds exactly like $\mathcal{C}_0$ except for the challenge query. To answer the challenge query, Δ sends two random $\mathbb{G}$ elements as re-randomized group elements of the PS multi-signature in $\mathbf{gsk}[id_b]$, and programs oracle $\mathcal{H}_1$ to compute a proof of knowledge of sk_{id_b} and a'_b, and complete the challenge group signature.

To show that DGS satisfies anonymity, it suffices to show that $\mathcal{C}_0$ and $\mathcal{C}_1$ are both computationally indistinguishable from Δ under the first-group DDH and the SDL assumptions.

Assume that the SDL assumption holds over G and that there exists an efficient adversary $\mathcal{A}$ that can distinguish $\mathcal{C}_0$ from Δ with probability at least ε and in time $\tau_{\mathcal{A}}$. Assume that $p > 8q_{\mathcal{H}_1}/\varepsilon$. Consider an algorithm $\mathcal{B}$ that runs $\mathcal{A}$ as a subroutine and interacts with a DDH challenger $\mathcal{C}^{\mathrm{DDH}}_\beta$ that outputs a Diffie–Hellman tuple if $\beta = 1$ or a uniformly random tuple if $\beta = 0$.

Upon receiving the description of a pairing group $\Gamma = (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e)$ and of a tuple $(g_1, g_2, g_3, g_4) = \left(g_1, g_1^{\mu}, g_1^{\nu}, g_1^{\xi}\right)$ from the DDH challenger, $\mathcal{B}$ sets g_1 as the generator of $\mathbb{G}$ in the public parameters and generates the other DGS parameters.

$\mathcal{B}$ starts by executing the key-generation protocol with $\mathcal{A}$ and reconstructing the shares of the dishonest managers (which is possible since there are at least $t_I + 1$ honest issuers). Algorithm $\mathcal{B}$ then chooses two identities id_0^* and id_1^* uniformly at random. To answer game-oracle queries, $\mathcal{B}$ proceeds as algorithm $\mathcal{A}'$ in Lemma 2, except that everything done for id^* is duplicated for id_0^* and id_1^*, and except for GSign queries on $(id_d^*, *, *)$ for $d \in \{0, 1\}$, for all GOpen queries and for the challenge query. For those queries, $\mathcal{B}$ proceeds as follows:

- $\mathsf{GSign}(id_d^*, m)$: check that $id^* \in Q_{\mathsf{GJoin}} \setminus Q_{\mathsf{Corrupt}}$. Fetch $(\perp, ({a'_d}^*, \Sigma^*_{d,1}, \Sigma^*_{d,2})) \leftarrow \mathbf{gsk}[id_d^*]$, generate $\alpha \in_R \mathbb{Z}_p^*$, compute $(\Sigma'_1, \Sigma'_2) \leftarrow (\Sigma^*_{d,1}, \Sigma^*_{d,2})^{\alpha}$, simulate a proof π of knowledge of $\mu sk_{id_d^*}$ and ${a'_d}^*$ by programming $\mathcal{H}_1$. Add $(id, m, \sigma = (\Sigma'_1, \Sigma'_2, \pi), \alpha)$ to Q_{Sign} and return σ. The random α used to compute the signature is necessary to later correctly simulate the opening of group signatures as $\mathcal{B}$ does not have access to $\tilde{g}_2^{\mu}$ as in the previous lemma.
- $\mathsf{GOpen}_i(O, m, \sigma = (\Sigma_1, \Sigma_2, \pi))$: check that $i \in O \in \binom{[n_O]}{t_O+1}$. Add (m, σ) to Q_{GOpen}. A GOpen execution is triggered. Play the role of the honest openers in I. Check that $\mathsf{GVerf}(ipk, m, \sigma) = 1$. By Lemma 2, this query cannot be a forgery for id_0^* or id_1^* (in the event in which $\mathcal{A}$ queries Ch on $(id_0^*, id_1^*, *)$

and receives an answer different from $\perp$). For all (id, m, σ, α) in Q_{Sign}, test the equality

$$e\left(\Sigma_1, \tilde{g}_1^{x+y_1 a'^*}\right) e\left(g_2^{r_d^* \alpha}, \tilde{Y}_0^{sk_{id_d^*}}\right) = e(\Sigma_2, \tilde{g}_1).$$

for all $d \in \{0,1\}$. If it holds for some $d \in \{0,1\}$ and if $(id_d^*, m, *) \in Q_{\mathsf{GSign}}$ (which implies $id_d^* \in Q_{\mathsf{GJoin}}$) then fetch $\mathbf{reg}_i[id_d^*] = (*, (s_{d,i})_{i=1}^{n_O})$. For $j \in I \setminus \{i\}$, compute $T_{id_d^*,j} \leftarrow e(\Sigma_1, \tilde{Y}_0)^{s_{d,j}}$. Compute

$$T_{id_d^*,i} \leftarrow \left(e\left(g_2^{r_d^* \alpha sk_{id_d^*}}, \tilde{Y}_0\right) \prod_{j \in I \setminus \{i\}} T_{id_d^*,j}^{-w_j}\right)^{1/w_i}$$
$$= e(\Sigma_1, \tilde{Y}_0)^{\left(\mu sk_{id_d^*} - \sum_{j \neq i} s_j w_j\right)/w_i}.$$

The values $(T_{id_d^*,j})_{j \in I}$ are indistinguishable from the values in the real protocol (in which case it would be $sk_{id_d^*}$ instead of $\mu sk_{id_d^*}$) as $\mathcal{A}$ never receives g_2. For the identities $id \neq id_d^*$ such that $\mathbf{reg}_i[id] \neq \perp$, compute the $T_{id,i}$ values as in the scheme. Continue as in Lemma 2.

– For the challenge query, if $(id_0, id_1) \neq (id_0^*, id_1^*)$ then halt the interaction with $\mathcal{A}$ and send 0 to the DDH challenger. Check that id_0^* and id_1^* are in $Q_{\mathsf{GJoin}} \setminus Q_{\mathsf{Corrupt}}$, that $\mathbf{gsk}[id_d^*] \neq \perp$ for both $d \in \{0,1\}$ and that $(*, m^*, \sigma^*) \notin Q_{\mathsf{GSign}}$. Set $\Sigma_1 \leftarrow g_3$ and $\Sigma_2 \leftarrow g_3^{x+y_1 a_0'^*} g_4^{sk_{id_0^*}}$. Compute a simulated proof π of knowledge of $\mu sk_{id_0^*}$ and $a_0'^*$ by programming $\mathcal{H}_1$. Return $\sigma \leftarrow (\Sigma_1, \Sigma_2, \pi)$. Note that if $\beta = 1$, then the σ has the same distribution as the challenge signature computed by $\mathcal{C}_0$, whereas in case $\beta = 0$, Σ_2 is uniformly random in $\mathbb{G}^*$, and therefore σ has the same distribution as the signature computed by Δ.

If $\beta = 1$ then algorithm $\mathcal{B}$ is perfectly indistinguishable from $\mathcal{C}_0$, and if $\beta = 0$ then it is perfectly indistinguishable from Δ. The advantage of $\mathcal{B}$ in the DDH game is then at least ε.

It follows that

$$\varepsilon \leq |ID|^2 \mathbf{Adv}^{\mathrm{DDH}-1}_{\mathsf{G},\mathcal{B}(\mathcal{A})}(\lambda) + \frac{8p}{p-1}|ID|\mathbf{Adv}^{\mathrm{SDL}}_{\mathsf{G},\mathcal{B}(\mathcal{A})}(\lambda)$$
$$+ \frac{|ID|(q_{\mathcal{H}_1}+1)}{p} + n_O|ID|\mathbf{Adv}^{\mathrm{DDH}-2}_{\mathsf{G},\lambda}(\mathcal{B}(\mathcal{A})).$$

Similarly, if $\mathcal{A}$ can distinguish Δ from $\mathcal{C}_1$, then $\mathcal{B}$ (by using $\mathbf{gsk}[id_1^*]$ instead of $\mathbf{gsk}[id_0^*]$ for the challenge query) wins the DDH game with probability at least ε.

Therefore,

$$\mathbf{Adv}^{\mathrm{ano}}_{\mathsf{G},\mathsf{PS\text{-}DGS},n_I,n_O,t_I,t_O,\mathcal{A}}(\lambda) \leq 2\Big(|ID|^2 \mathbf{Adv}^{\mathrm{DDH}-1}_{\mathsf{G},\mathcal{B}(\mathcal{A})}(\lambda) + \frac{8p}{p-1}|ID|\mathbf{Adv}^{\mathrm{SDL}}_{\mathsf{G},\mathcal{B}(\mathcal{A})}(\lambda)$$
$$+ \frac{|ID|(q_{\mathcal{H}_1}+1)}{p} + n_O|ID|\mathbf{Adv}^{\mathrm{DDH}-2}_{\mathsf{G},\lambda}(\mathcal{B}(\mathcal{A}))\Big)$$

and the theorem follows. □

Proof (of Theorem 3 – Traceability).

The proof consists in reducing the traceability of DGS to the existential unforgeability of the PS signature scheme. As its unforgeability relies on the $q_{\mathcal{H}_0}$-MSDH-1 assumption, the theorem follows. (Notice that GJoin.U, GJoin.I$_i$ and GOpen queries induce $\mathcal{H}_0$ queries.)

The proof idea is to apply the forking algorithm of the generalized forking lemma (Lemma 5) to an algorithm $\mathcal{A}'$ that runs $\mathcal{A}$ as a subroutine in order to obtain two distinct group signatures from which a PS signature forgery can be computed.

Suppose that there exists an adversary $\mathcal{A}$ that wins the $\binom{n_I,n_O}{t_I,t_I}$ traceability game for DGS with probability at least ε and in time $\tau_{\mathcal{A}}$. Further assume that $p > 8q_{\mathcal{H}_1}/\varepsilon$.

We first define $\mathcal{A}'$ and then a reduction algorithm $\mathcal{B}$ that interacts with the forgery-game challenger $\mathcal{C}$ for PS and applies the general forking lemma to $\mathcal{A}'$ to win the forgery game.

$\mathcal{A}'$ is an algorithm that runs on the input of public parameters pp as in the scheme, of a public key $(\tilde{X}, \tilde{Y}_0, \tilde{Y}_1)$ for the PS signature scheme and of a random tape ρ.

Throughout its interaction with $\mathcal{A}$, algorithm $\mathcal{A}'$ maintains the same global variables as the challenger of the traceability game. It initializes a counter $ctr \leftarrow 0$ and sets $Q_{\mathcal{H}_1} \leftarrow \emptyset$.

Throughout the experiment, $\mathcal{A}'$ answers random-oracle queries as follows:

- $\mathcal{H}_0(id)$: forward the query to $\mathcal{B}$.
- $\mathcal{H}_1(b \in \{0,1\}^*)$: if $(b, *) \notin Q_{\mathcal{H}_1}$ then generate $c \in_R \mathbb{Z}_p$, do $ctr \leftarrow ctr + 1$, add (b, c, ctr) to $Q_{\mathcal{H}_1}$ and return c; else retrieve (b, c, ι) from $Q_{\mathcal{H}_1}$ and return c.

At the beginning of the experiment, $\mathcal{A}'$ runs three times the simulator of key-generation protocol of Gennaro et al. with $\mathcal{A}$ on each $\tilde{X}$, $\tilde{Y}_0$ and $\tilde{Y}_1$ respectively, and plays the role of the honest issuers. At the end of the protocol executions, all honest issuers return an issuer public key ipk. Moreover, as there are at least $t_I + 1$ honest issuers, so $\mathcal{A}'$ can reconstruct the share x_i, $y_{i,0}$ and $y_{i,1}$ of the dishonest issuers. Let $\tilde{X}'$, $\tilde{Y}'_0$ and $\tilde{Y}'_1$ be the group elements defined by the shares generated during the simulation, i.e., $\tilde{X}' = \tilde{g}^{\sum_{i \in I} x_i w_i}$ (and $x' \coloneqq \sum_{i \in I} x_i w_i$) for any set $I \in \binom{[n_I]}{t_I+1}$, where w_i is the Lagrange interpolation coefficient at position i; and similarly for $\tilde{Y}'_0$ and $\tilde{Y}'_1$.

For each of the honest openers, $\mathcal{A}'$ generates a pair of keys as in the scheme.

Algorithm $\mathcal{A}'$ then answers oracles queries as follows (if a check fails or if it ever aborts, $\mathcal{A}'$ returns $(\emptyset, \emptyset, \emptyset)$):

- GJoin.U(id, I) : check that $I \in \binom{[n_I]}{t_I+1}$. Add id to Q_{GJoin}. A GJoin execution is triggered. Play the role of the user and of the honest issuers with the shares generated by the simulator of the distributed key-generation protocol.

Upon receiving signature shares $\Sigma_{2,i}$ from each of the corrupt issuers played by the adversary, test whether the equality

$$e\left(\Sigma_1, \tilde{X}' \tilde{Y}_0'^{sk_{id}} \tilde{Y}_1'^{a'}\right) = e\left(\prod_{i\in I} \Sigma_{i,2}^{w_i}, \tilde{g}\right)$$

holds, i.e., test whether the signature is valid for the key $(\tilde{X}', \tilde{Y}_0', \tilde{Y}_1')$. If not, abort the protocol, otherwise make a signing query to $\mathcal{C}$ on sk and receive a signature $\Sigma = (a', \Sigma_1, \Sigma_2)$. Note that $\Sigma_2 \neq \prod_{i\in I} \Sigma_{2,i}^{w_i}$ with overwhelming probability, because of the simulator of the key-generation protocol. Follow the joining protocol and set $\mathbf{gsk}[id] \leftarrow (sk, \Sigma)$. Note also that even if $\mathcal{A}$ causes the protocol to abort, or delays or drops messages, $\mathcal{B}$ has already added to Q_{GJoin} at the beginning of the protocol.

– $\mathsf{GJoin.I}_i(id, I)$: check that $i \in I \in \binom{[n_I]}{t_I+1}$. Adds id to Q_{GJoin} and Q_{Corrupt}. A GJoin protocol execution is triggered. Play the role of the honest issuers.
Upon receiving $\mathsf{written}$ from the user played by $\mathcal{A}$, parse $L[id]$ as in the protocol. Make an internal query $(a', h) \leftarrow \mathcal{H}_0(id)$. Perform the same verifications as in the real scheme for each of the honest issuers.
If the verification of the proof on the encrypted shares holds but that the shares are not actually valid i.e., the soundness of the NIZK proof was broken, abort the protocol and return $(\emptyset, \emptyset, \emptyset)$. Note since there are at least $t_O + 1$ honest openers, $\mathcal{A}'$ can test whether the shares are valid as it then has enough shares to recompute the user polynomial.
If the verification $\mathsf{NIZK.Verf}(g, h, g_{sk}, h_{sk}, \pi) \stackrel{?}{=} 1$ succeeds, then (g, g_{sk}, h, h_{sk}) is a Diffie–Hellman tuple. Run $sk_{id} \leftarrow \mathsf{Ext}(\Gamma, g, g_{sk}, h, h_{sk})$. Abort the interaction with $\mathcal{A}$ if the extraction fails. Unless $\mathcal{A}$ can solve the discrete-logarithm problem in $\mathbb{G}$, which occurs only with negligible probability under the SDL assumption over G (which is itself implied by the q-MSDH-1 assumption), $sk_{id} = \mathrm{dlog}_g(g_{sk}) = \mathrm{dlog}_h(h_{sk})$. If the equality does not hold, then abort the interaction with $\mathcal{A}$. Otherwise, make a signing query on sk_{id} to $\mathcal{C}$ and receive a signature $\Sigma = (a', \Sigma_1 = h, \Sigma_2)$.
For the honest issuers $j \neq i$, compute $\Sigma_{2,j} \leftarrow h^{x_j + y_{j,0} sk_{id} + y_{j,1} a'}$.
For issuer i, compute

$$\begin{aligned}\Sigma_{2,i} &\leftarrow \left(\Sigma_2 \prod_{j\in I\setminus\{i\}} \Sigma_{2,j}^{-w_j}\right)^{1/w_i} \\ &= h^{\left(x - x' + (y_0 - y_0')sk_{id} + (y_1 - y_1')a'\right)/w_i + x_i + y_{i,0} sk_{id} + y_{i,1} a'}.\end{aligned}$$

Follow the rest of the protocol.
Note that even if $\mathcal{A}$ causes the protocol to abort, or delays or drops messages, $\mathcal{A}'$ has already added to Q_{GJoin} and Q_{Corrupt} at the beginning of the protocol. It implies that if $\mathcal{B}$ has made a signing query on an identity id, then $id \in Q_{\mathsf{GJoin}}$.

– $\mathsf{GSign}(id, m)$: check that $id \in Q_{\mathsf{GJoin}} \setminus Q_{\mathsf{Corrupt}}$. Compute $\sigma \leftarrow \mathsf{GSign}(ipk, \mathbf{gsk}[id], m)$. Add (id, m, σ) to Q_{GSign}. Return σ.

– $\mathsf{GOpen}_i(O, m, \sigma)$: (for $i \in \mathcal{HM}$) check that $i \in O \in \binom{[n_O]}{t_O+1}$. Add (m, σ) to Q_{GOpen}. Play the role of the honest openers in I and follow the protocol.
– $\mathsf{RevealU}(id)$: add id to Q_{RevealU}. Returns $\mathbf{gsk}[id]$.
– $\mathsf{ReadReg}(i, id)$: return $\mathbf{reg}_i[id]$.

In the event in which $\mathcal{A}$ wins the game, it ultimately outputs a tuple (O^*, m^*, σ^*). Algorithm $\mathcal{A}'$ verifies that $O^* \notin \binom{\{t_O^*+1,\ldots,n_O\}}{t_O+1}$ and that $\mathsf{GVerf}(ipk, m^*, \sigma^*) = 1$. If $\mathcal{A}$ does not output such a tuple or if the verifications fail, then $\mathcal{A}'$ returns $(\emptyset, \emptyset, \emptyset)$, otherwise it runs

$$\langle \{id_i^*\}_{i \in O^*} \rangle \leftarrow \langle \{\mathsf{GOpen}(\mathbf{reg}_i, osk_i, O^*, gpk, m^*, \sigma^*)\}_{i \in O^*} \rangle.$$

By Lemma 1, for all $i, j \in O^*$, $id_i^* = id_j^*$.

Moreover, since for each honest opener, any id for which $\mathbf{reg}_i[id] \neq \perp$ is such that $id \in Q_{\mathsf{GJoin}}$ (for $\mathcal{A}'$ starts by adding to Q_{GJoin} the identity of any joining query), it is impossible that the opening algorithms return an identity that is not in Q_{GJoin}. That is, setting $id^* \leftarrow \max O^*$, if $id^* \neq \perp$, then $id^* \in Q_{\mathsf{GJoin}}$ necessarily.

The winning conditions thus imply that in the event in which $\mathcal{A}$ wins the traceability game, either $id^* = \perp$, or $id^* \in Q_{\mathsf{GJoin}} \setminus Q_{\mathsf{Corrupt}}$ and $(id^*, m^*, \sigma^*) \notin Q_{\mathsf{GSign}}$.

By Lemma 2, $(id^*, m^*, \sigma^*) \notin Q_{\mathsf{GSign}}$, i.e., the second case occurs with negligible probability under the SDL assumption.

If $\mathcal{A}$ did not win the traceability game, $\mathcal{A}'$ returns $(\emptyset, \emptyset, \emptyset)$. Otherwise, parsing σ^* as $(\Sigma_1^*, \Sigma_2^*, \pi^* = (c^*, v_{sk}^*, v_{a'}^*))$, let ι_{σ^*} be the computation step at which $\mathcal{A}$ queried $\mathcal{H}_1$ on $b_{\sigma^*} \leftarrow (ipk, \Sigma_1^*, \Sigma_2^*, u^*, m^*)$ with

$$u^* := e\left(\Sigma_1^{*\,v_{sk}^*}, \tilde{Y}_0\right) e\left(\Sigma_1^{*\,v_{a'}^*}, \tilde{Y}_1\right) e\left(\Sigma_1^{*\,-c}, \tilde{X}\right) e\left(\Sigma_2^{*\,c}, \tilde{g}_1\right),$$

be it the step at which it outputs the forgery, i.e., $(b_{\sigma^*}, c^*, \iota_{\sigma^*}) \in Q_{\mathcal{H}_1}$. Algorithm $\mathcal{A}'$ returns $(\{\iota_{\sigma^*}\}, \sigma^*, \emptyset)$.

Set

$$\tilde{\varepsilon} \leftarrow \varepsilon - |ID| \left(\mathbf{Adv}_{\mathsf{G},\lambda}^{\mathrm{ADH-KE}}(\mathcal{B}(\mathcal{A})) + \mathbf{Adv}_{\mathsf{G},\lambda}^{\mathrm{DLOG-1}}(\mathcal{B}(\mathcal{A}))\right)$$
$$- \frac{8p}{p-1}|ID|\mathbf{Adv}_{\mathsf{G},\mathcal{B}(\mathcal{A})}^{\mathrm{SDL}}(\lambda) - \frac{|ID|(q_{\mathcal{H}_1}+1)}{p} - n_O|ID|\mathbf{Adv}_{\mathsf{G},\lambda}^{\mathrm{DDH-2}}(\mathcal{B}(\mathcal{A})).$$

It is a lower bound on the probability that $\mathcal{A}$ wins the traceability game for DGS, that $\mathcal{A}'$ never aborts during GJoin execution (prompted by a $\mathsf{GJoin.I}_i$ query) due to extraction failure of a user secret key and that the identity output by the honest openers at the forgery phase is $\perp$.

Let then $\mathcal{B}$ be an algorithm that runs $\mathcal{A}'$ as subroutine and interacts with the forgery-game challenger $\mathcal{C}$ for PS.

Upon receiving pp_{PS} and vk from the challenger, $\mathcal{B}$ sets generates $g \in_R \mathbb{G}^*$ and sets $pp \leftarrow (pp_{\mathsf{PS}}, g, n_I, n_O, t_I, t_O)$. It then runs $\mathcal{A}'$ on the input of pp, $(\tilde{X}, \tilde{Y}_0, \tilde{Y}_1)$ and a uniformly random tape ρ.

Throughout its interaction with $\mathcal{A}'$, algorithm $\mathcal{B}$ answers random-oracle queries as follows:

– $\mathcal{H}_0(id)$: if $(id, *) \notin Q_{\mathcal{H}_0}$ then forward the query to $\mathcal{C}$, receive (a', h), add $(id, (a', h))$ to $Q_{\mathcal{H}_0}$ and return (a', h); else retrieve $(id, (a', h))$ from $Q_{\mathcal{H}_0}$ and return (a', h).

Recall that $p > 8tq_{\mathcal{H}_1}/\varepsilon$ by assumption. With probability at least $\tilde{\varepsilon}/8$ and with running time at most $8q_{\mathcal{H}_1}/\tilde{\varepsilon} \cdot \ln(8/\tilde{\varepsilon}) \cdot \tau_{\mathcal{A}}$, the forking algorithm of the generalized forking lemma (Lemma 5) applied to $\mathcal{A}'$ returns $(\{\iota_{\sigma^*}\}, \{\sigma^*\}, \{\sigma^{*\prime}\})$ with σ^* and $\sigma^{*\prime}$ as group-signature forgeries that open to $\perp$. The lemma ensures that for $\sigma^* = (\Sigma_1^*, \Sigma_2^*, \pi^* = (c^*, v_{sk}^*, v_{a'}^*))$ and $\sigma^{*\prime}$ similarly parsed, the computation step at which the challenges c^* and $c^{*\prime}$ were computed are the same, and those challenge are distinct modulo p.

As the inputs to $\mathcal{A}$ and its randomness are identical until that $\mathcal{H}_1$ query, $\Sigma_1^* = \Sigma_1^{*\prime}$, $\Sigma_2^* = \Sigma_2^{*\prime}$, $u^* = u^{*\prime}$ and $m^* = m^{*\prime}$ necessarily. It follows that

$$e\left(\Sigma_1^*, \tilde{Y}_0^{(v_{sk}^* - v_{sk}^{*\prime})/(c^{*\prime} - c^*)} \tilde{Y}_1^{(v_{a'}^* - v_{a'}^{*\prime})/(c^{*\prime} - c^*)}\right) = e(\Sigma_2^*, \tilde{g}_1) e(\Sigma_1^*, \tilde{X})^{-1}.$$

Set $sk^* \leftarrow (v_{sk}^* - v_{sk}^{*\prime})/(c^{*\prime} - c^*)$, $a'^* \leftarrow (v_{a'}^* - v_{a'}^{*\prime})/(c^{*\prime} - c^*)$ and $\Sigma^* \leftarrow (a'^*, \Sigma_1^*, \Sigma_2^*)$. The latter is then a valid multi-signature on sk^*.

However, no honest issuer ever signed sk^* as there would otherwise exist an identity $id^* \neq \perp$ such that $\mathbf{reg}_i[id^*] = s_i^*$ for all opener $i \in O^*$ and $sk^* = \sum_{i \in I^*} s_i^* w_i$. It is the case since in each execution of GOpen, the openers always start by updating their registers. That identity id^* would have then be returned by the honest openers.

Algorithm $\mathcal{B}$ then sends sk^* and Σ^* to $\mathcal{C}$ as forgery and wins the existential forgery game for PS with probability at least $\tilde{\varepsilon}/8$. Therefore,

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{trace}}_{\mathsf{G},\mathsf{DGS},N,t,\mathcal{A}}(\lambda) \leq\ & 8\mathbf{Adv}^{\mathrm{EUF-CMA}}_{\mathsf{G},\mathsf{PS},1,\mathcal{A}}(\lambda) \\
& + |ID| \left(\mathbf{Adv}^{\mathrm{ADH-KE}}_{\mathsf{G},\mathcal{B}(\mathcal{A})}(\lambda) + \mathbf{Adv}^{\mathrm{DLOG}-1}_{\mathsf{G},\mathcal{B}(\mathcal{A})}(\lambda)\right) \\
& + \frac{8p}{p-1}|ID|\mathbf{Adv}^{\mathrm{SDL}}_{\mathsf{G},\mathcal{B}(\mathcal{A})}(\lambda) + \frac{|ID|(q_{\mathcal{H}_1}+1)}{p} \\
& + n_O|ID|\mathbf{Adv}^{\mathrm{DDH}-2}_{\mathsf{G},\lambda}(\mathcal{B}(\mathcal{A}))
\end{aligned}$$

As the first-group discrete logarithm and the SDL assumptions are both implied by the $q_{\mathcal{H}_0}$-MSDH-1 assumption, the theorem follows. □

C Threshold Group Signatures without Ledger

The construction in Section 4 requires an append-only ledger for users to communicate their secret-key shares to the openers. Such ledgers can be implemented in practice, but it is yet an additional assumption. We therefore also propose a scheme that does not require a ledger. However, this comes for the price of combining the roles of issuer and opener, as it is the case the case for the original GetShorty scheme of Bichsel et al. [8]. The authorities are now referred to as managers, and suppose that there are n of them.

For the sake of simplicity, assume now that the issuance and opening thresholds are the same and denote it t. However, instead of having the same threshold for corruption, one could define (t, t_c)-out-of-n threshold group signatures as group signatures with n managers, of which $t+1$ must collaborate to add a user or to open a signature, and of which at most t_c can be corrupt. Defining a separate corruption threshold gives the flexibility to corrupt some managers, but not to many so that any two sets of t have at least one honest manager in common. This property ensure that any manager who did not add a user can recover her secret-key shares from an honest manager who added her when a signature is to be opened.

In this model, the main changes from the previous construction are as follows.

Key Generation. The managers now simply generate ElGamal keys separately and run the distributed key-generation protocol of Gennaro et al. [32] to generate PS keys. They use t as reconstruction threshold and t_c as corruption threshold.

Note that the original protocol of Gennaro et al. does not distinguish the reconstruction from the corruption threshold. However, their protocol can be adapted to a setting with two thresholds. It remains secure and robust (i.e., it terminates) as long as the number of honest parties (at least $n - t_c$) is greater than the reconstruction threshold (t in the present case), so that the honest parties can reconstruct the shares of qualified participants who received a valid complaint during the extraction phase [32, Fig. 2]. The two thresholds t and t_c must only satisfy $n - t_c > t$.

Join. As before, the user encrypts a t-out-of-n Shamir share of sk_{id} for each manager, even for the non-participating ones, and proves the validity of the ciphertexts. Each participating manager then verifies the correctness of the proofs. The difficulty is to ensure that the non-participating managers indeed get those ciphertexts now that there is no ledger.

Assuming $t \geq (n + t_c - 1)/2$ ($\geq t_c$), any two manager sets I and J of size $t+1$ have an intersection $I \cap J$ of size $|I|+|J|-|I \cup J| \geq 2(t+1)-n \geq t_c+1$. It means that for every group member, any set of $t+1$ managers will always contain at least one honest manager that added her.

To ensure that the non-participating managers can later recover the shares, it suffices to have each manager in the joining protocol sign the encrypted shares of the user secret key with a multi-signature scheme (BLS [12, Section 3.1] for instance) after verifying the shares. Each manager then saves the shares and their multi-signatures in registers.

Update Registers. Afterwards, any set of $t+1$ managers can synchronize their registers and retrieve shares of every group-member secret key by first broadcasting the list of users they added. Next, turn by turn in lexicographic order, they broadcast the encrypted shares and multi-signatures for each user such that there is a manager who did not add her, and for which shares with a valid multi-signature have not not been broadcast yet.

Since $t \geq t_c$, a valid multi-signature by $t+1$ managers on a set of shares implies that at least one honest manager has signed them (after verifying

them), so they are authentic. The managers can then update their registers by decrypting the shares they receive. Moreover, for every group member, the set of $t+1$ managers that added her has at least t_c+1 managers in common with the $t+1$ managers synchronizing their registers. Therefore, at least one honest manager will broadcast valid shares of her secret key, and each manager is guaranteed that his register now contains a valid secret-key share for every user that was ever added.

Sign&Verify. Computing and verifying signatures are done as in the Section 4 construction.

Open. As for opening, after synchronizing their registers, the managers participating in the opening protocol proceed as before.

Overall, the scheme is secure if $n - t_c > t \geq (n + t_c - 1)/2$ (which implies $t_c < \min(t, n/3)$). An interpretation of these bounds is that the joining threshold should be large enough so that for every group member, any set of openers contains at least one honest manager who added her, but not too large for the honest managers to be able to securely generate issuance keys.

D Multi-Signatures with Key Aggregation

A multi-signature scheme [43] allows a number $n \geq 1$ of signers to jointly compute a short signature on the same message. Given the public verification keys of the n signers, one can verify that all n signers signed the message.

Syntax. Formally, for an integer $n > 0$, an n-signature scheme MS (with key aggregation) consists of the following algorithms:

$\mathsf{MS.Setup}(1^\lambda, n, aux) \to pp$: returns public parameters on the input of a security parameter, a number of signers and an auxiliary input.

$\mathsf{MS.KG}(pp) \to (vk, sk)$: returns a pair of verification–signing keys on the input of public parameters.

$\mathsf{MS.KAggreg}(vk_1, \ldots, vk_n) \to avk$: aggregates the verification keys of n signers and returns a short aggregated key avk that can be used to verify aggregated signatures.

$\mathsf{MS.Sign}(sk, m) \to \sigma$: a signing algorithm which takes as an input a signing key sk and a message m. It returns a signature σ.

$\mathsf{MS.SAggreg}((vk_i)_{i=1}^n, m, (\sigma_i)_{i=1}^n) \to \sigma$: an algorithm that aggregates the signatures on a single message m computed by n signers and returns a short aggregated signature σ.

$\mathsf{MS.Verf}(avk, m, \sigma) \to b \in \{0, 1\}$: on the input of an aggregated verification key, a message and an aggregated signature, returns a bit indicating whether the signature is valid w.r.t. the aggregated verification key.

An alternative definition in which the verification keys are aggregated during signature verification could be considered but would be less efficient in a setting in which the set of signers is fixed (or at least rarely changes). Indeed, if the set of signers is fixed, their keys can be aggregated once for all and the resulting aggregated key can be used every time a signature is to be verified.

Security Model. The security of an n-signature scheme [5] MS is defined via a security game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. At the beginning of the game, $\mathcal{C}$ generates parameters $pp \leftarrow \mathsf{MS.Setup}(1^\lambda, n, aux)$ and sends them to $\mathcal{A}$. Adversary $\mathcal{A}$ then sends a target honest-signer identity i^* to $\mathcal{C}$. Challenger $\mathcal{C}$ proceeds by generating keys $(vk_{i^*}, sk_{i^*}) \leftarrow \mathsf{MS.KG}(pp)$ and sending vk_{i^*} to $\mathcal{A}$. Adversary $\mathcal{A}$ is now allowed to issue signing queries on arbitrary messages m. To answer such a query, $\mathcal{C}$ computes and sends $\sigma_{i^*} \leftarrow \mathsf{MS.Sign}\,(sk_{i^*}, m)$ to $\mathcal{A}$. After the query phase, $\mathcal{A}$ outputs a set of verification keys K such that $vk_{i^*} \in K$, a message m for which no signing query has been made and a signature σ. Adversary $\mathcal{A}$ wins the game if and only if $\mathsf{MS.Verf}(\mathsf{MS.KAggreg}(K), m, \sigma) = 1$. A multi-signature scheme is existentially unforgeable under chosen-message attacks (or EUF-CMA secure) if no efficient adversary can win this security game with a non-negligible probability.

A weak unforgeability can also be defined via a variant of the previous game in which $\mathcal{A}$ outputs, along with index i^* (so before getting key vk_{i^*}), a list of messages that $\mathcal{C}$ signs with key sk_{i^*} and sends back the results with vk_{i^*}. Adversary $\mathcal{A}$ cannot make signing queries afterwards. Scheme MS is weakly existentially unforgeable (or EUF-wCMA secure) if no efficient adversary $\mathcal{A}$ has a non-negligible probability of winning this variant of the security game.

E PS Multi-Signatures

This section presents the original PS signature scheme and then proves the unforgeability of the PS multi-signature scheme in Section 5.1.

E.1 Original Pointcheval–Sanders Signature Scheme

Pointcheval and Sanders [46] introduced a single-message signature scheme that they proved [47, Section 5.1] to be weakly existentially unforgeable under chosen-message attacks (or EUF-wCMA secure) under the q-MDSH-1 assumption (Section 1). It is actually also existentially unforgeable under chosen-message attacks (EUF-CMA) under an interactive assumption [46, Assumption 2] that Pointcheval and Sanders proved to hold in thegeneric group model.

Given a type-3 pairing-group generator G and a security parameter $\lambda \in \mathbb{N}$, the Pointcheval–Sanders (PS) signature scheme in a pairing group $\Gamma = \big(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e\big) \leftarrow \mathsf{G}(1^\lambda)$ consists of the following algorithms:

$\mathsf{PS.Setup}(1^\lambda, \Gamma) \to pp$: generate $\tilde{g} \in_R \mathbb{G}^*$. Return $pp \leftarrow (\Gamma, \tilde{g})$.
$\mathsf{PS.KG}(pp) \to (vk, sk)$: generate $x, y \in_R \mathbb{Z}_p^*$, compute $\tilde{X} \leftarrow \tilde{g}^x$, $\tilde{Y} \leftarrow \tilde{g}^y$, and set $vk \leftarrow \Big(\tilde{g}, \tilde{X}, \tilde{Y}\Big)$ and $sk \leftarrow (x, y)$. Return (vk, sk).
$\mathsf{PS.Sign}(sk, m) \to \sigma$: choose $h \in_R \mathbb{G}^*$ and return $\sigma \leftarrow (h, h^{x+ym})$;
$\mathsf{PS.Verf}(vk, m, \sigma) \to b$: parse σ as (σ_1, σ_2), verify that $\sigma_1 \neq 1_{\mathbb{G}_1}$ and that $e\big(\sigma_1, \tilde{X}\,\tilde{Y}^m\big) = e(\sigma_2, \tilde{g})$. If so, return 1, otherwise return 0.

This scheme can be extended to a multi-message version [46, Section 4.2] as follows. To sign a tuple of messages $(m_1, \ldots, m_k) \in \mathbb{Z}_p^k$ with a signing key $(x, y_1, \ldots, y_k)$ for which the corresponding verification key is $\Big(\tilde{g}, \tilde{X} = \tilde{g}^x, \tilde{Y}_1 = \tilde{g}^{y_1}, \ldots, \tilde{Y}_k = \tilde{g}^{y_k}\Big)$, the signing algorithm generates $h \in_R \mathbb{G}^*$ and returns $\sigma \leftarrow \big(h, h^{x+\sum y_j m_j}\big)$. The verification algorithm then checks that $\sigma_1 \neq 1_{\mathbb{G}}$ and that $e\big(\sigma_1, \tilde{X} \prod \tilde{Y}_j^{m_j}\big) = e\big(\sigma_2, \tilde{g}\big)$.

E.2 Rewinding Lemma

Applied to an algorithm, the rewinding lemma [15, Variant of Lemma 19.2] gives a lower-bound on the probability of forking the algorithm at a certain step of its computation. This lemma provides a crucial argument in the proof of the weak unforgeability of the original PS single-message multi-signature scheme (Lemma 3).

Lemma 6 (Rewinding Lemma). *Let S, Θ and T be non-empty finite sets. Consider a function $f\colon S \times \Theta \times T \to \{0,1\}$. Let X, Y, Y', Z and Z' be mutually independent random variables such that X takes values in S, Y and Y' take values in Θ, and Z and Z' take values in T. Setting $\varepsilon = \Pr[f(X,Y,Z) = 1]$,*

$$\Pr[f(X,Y,Z) = 1, f(X,Y',Z') = 1, Y \neq Y'] \geq \varepsilon^2 - \varepsilon/|\Theta|.$$

E.3 Unforgeability Proof

This section proves the two core lemmas of the unforgeability proof of PS multi-signatures.

Proof (of Lemma 3). For $\lambda \in \mathbb{N}$, consider a pairing group $\Gamma \leftarrow \mathsf{G}(1^\lambda)$. Suppose that there exists an efficient adversary $\mathcal{A}$ that wins the weak existential unforgeability game with at most one query to oracle $\mathcal{H}_1$ (the general case will be considered further) with probability at least ε. Consider a reduction algorithm $\mathcal{B}$ which runs $\mathcal{A}$ as a subroutine and receives a q-MSDH-1 instance from a challenger, i.e., receives two tuples $\Big(g^{x^\ell}, \tilde{g}^{x^\ell}\Big)_{\ell=0}^{q} \in (\mathbb{G}^*)^{q+1}$ and $(g^a, \tilde{g}^a, \tilde{g}^{ax}) \in \mathbb{G}^* \times (\tilde{\mathbb{G}}^*)^2$. Algorithm $\mathcal{B}$ has to output a tuple $\big(w, P, h^{1/x+w}, h^{a/P(x)}\big)$ with $h \in_R \mathbb{G}^*$, P a polynomial in $\mathbb{Z}_p[X]$ of degree at most q and $w \in \mathbb{Z}_p$ such that the polynomials $X + w$ and P are coprime. At the beginning of the weak unforgeability game, adversary $\mathcal{A}$ sends to $\mathcal{B}$ a tuple of messages $(w_1, \ldots, w_q) \in \mathbb{Z}_p^q$ for which it expects signatures. $\mathcal{B}$ computes group elements $G \leftarrow g^{\prod_{\ell=1}^q (x+w_\ell)}$ and $\tilde{G} \leftarrow \tilde{g}^{\prod_{\ell=1}^q (x+w_\ell)}$, and sends $(G, \tilde{G}, n, 1)$ to adversary $\mathcal{A}$ as public parameters. $\mathcal{A}$ sends the index $i^* \in [n]$ of a target (honest) signer. $\mathcal{B}$ sets $\tilde{X}_{i^*} \leftarrow \tilde{g}^{ax}$ and $\tilde{Y}_{i^*} \leftarrow \tilde{g}^a$, and outputs $\Big(\tilde{X}_{i^*}, \tilde{Y}_{i^*}\Big)$ as the verification key of signer i^*. Notice that this implicitly sets

$$x_{i^*} = \frac{ax}{\prod_{\ell=1}^q (x + w_\ell)}, \qquad y_{i^*} = \frac{a}{\prod_{\ell=1}^q (x + w_\ell)}.$$

For $\ell = 1, \ldots, q$, algorithm $\mathcal{B}$ computes a signature on w_ℓ as follows: it generates $t_\ell \in_R \mathbb{Z}_p^*$, and computes and stores $\sigma_\ell = (\sigma_{\ell,1}, \sigma_{\ell,2}) = \left(\left(g^{\prod_{r \neq l}(x+w_r)} \right)^{t_\ell}, (g^a)^{t_\ell} \right)$. Algorithm $\mathcal{B}$ can compute σ_ℓ from the q-MSDH-1 instance as the discrete logarithm of $\sigma_{\ell,1}$ to base g is a polynomial of degree $q-1$ in x and $\sigma_{\ell,2}$ is a single exponentiation of g^a. Note that

$$\begin{aligned} \sigma_{\ell,1}^{x_{i^*}+y_{i^*}w_\ell} &= \left(G^{\frac{t_\ell}{x+w_\ell}} \right)^{\frac{ax+aw_\ell}{\prod_{\ell=1}^{q}(x+w_\ell)}} = \left(G^{\frac{t_\ell}{x+w_\ell}} \right)^{\frac{a}{\prod_{r \neq l}(x+w_\ell)}} \\ &= g^{at_\ell} = \sigma_{\ell,2}, \end{aligned}$$

and thus σ_ℓ is a valid signature on w_ℓ for the verification key $\left(\tilde{X}_{i^*}, \tilde{Y}_{i^*} \right)$. Algorithm $\mathcal{B}$ then returns $\sigma_1, \ldots, \sigma_q$ to adversary $\mathcal{A}$.

Whenever adversary $\mathcal{A}$ queries random oracle $\mathcal{H}_0$ on a message m, algorithm $\mathcal{B}$ checks whether $m \in \{w_\ell\}_{\ell=1}^q$. If $m = w_\ell$ for some $\ell \in [q]$, algorithm $\mathcal{B}$ returns $\sigma_{\ell,1}$ as an answer to the random oracle query, otherwise if m has not been queried before, it generates, stores and returns $h \in_R \mathbb{G}^*$, otherwise (m is none of the w_ℓ and has been queried before), algorithm answers the query as it priorly did.

Whenever $\mathcal{A}$ makes its (unique) query to oracle $\mathcal{H}_1$ on $(vk_1, \ldots, vk_n) \in \tilde{\mathbb{G}}^{2n}$, algorithm $\mathcal{B}$ generates and returns $(t_1, \ldots, t_n) \in_R \Theta^n$. Adversary $\mathcal{A}$ eventually outputs a list of verification keys $\left(\tilde{X}_i, \tilde{Y}_i \right)_{i \neq i^*}$ and a valid forgery σ on a message $w \notin \{w_\ell\}_{\ell=1}^q$. If the unique query of $\mathcal{A}$ to $\mathcal{H}_1$ is not such that $vk_i = \left(\tilde{X}_i, \tilde{Y}_i \right)$ for $i = 1, \ldots, n$, then the forgery can be valid with probability at most $1/|\Theta|$, in which case algorithm $\mathcal{B}$ returns $\perp$ to the q-MSDH-1 challenger. Otherwise (the query to $\mathcal{H}_1$ is as such), as σ is a valid forgery, $\sigma_1 \neq 1_{\mathbb{G}}$ and

$$e\left(\sigma_1, \prod_{i=1}^{n} \tilde{X}_i^{t_i} \cdot \tilde{Y}_i^{wt_i} \right) = e\left(\sigma_1, \prod_{i \neq i^*} \tilde{X}_i^{t_i} \tilde{Y}_i^{wt_i} \right) e\left(\sigma_1, \tilde{X}_{i^*}^{t_{i^*}} \tilde{Y}_{i^*}^{wt_{i^*}} \right) = e\left(\sigma_2, \tilde{G} \right).$$

Algorithm $\mathcal{B}$ then rewinds $\mathcal{A}$ to the computation step at which it made its query to $\mathcal{H}_1$. Algorithm $\mathcal{B}$ generates $t'_{i^*} \in_R \Theta$ and replies with $(t_1, \ldots, t_{i^*-1}, t'_{i^*}, t_{i^*+1}, \ldots, t_n)$. Adversary $\mathcal{A}$ can make queries anew, and if it eventually outputs another forgery σ', then $\sigma'_1 \neq 1_{\mathbb{G}}$ and

$$e\left(\sigma'_1, \prod_{i \neq i^*} \tilde{X}_i^{t_i} \tilde{Y}_i^{wt_i} \right) e\left(\sigma'_1, \tilde{X}_{i^*}^{t'_{i^*}} \tilde{Y}_{i^*}^{wt'_{i^*}} \right) = e\left(\sigma'_2, \tilde{G} \right).$$

On that account,

$$e\left(\sigma_1'^{t'_{i^*}} / \sigma_1^{t_{i^*}}, \tilde{X}_{i^*} \tilde{Y}_{i^*}^{w} \right) = e\left(\sigma'_2 / \sigma_2, \tilde{G} \right),$$

so

$$e\left(\sigma_1'^{t'_{i^*}} / \sigma_1^{t_{i^*}}, \tilde{g}^{a(x+w)} \right) = e\left(\sigma'_2 / \sigma_2, \tilde{g}^{\prod_{\ell=1}^{q}(x+w_\ell)} \right).$$

It follows that $\left(\sigma_1'^{t'_{i^*}}/\sigma^{t_{i^*}}, \sigma_2'/\sigma_2\right)$ is then of the form $\left(h^{x+w}, h^{\frac{a}{\prod_{\ell=1}^{q}(x+w_\ell)}}\right)$.

Setting $P = \prod_{\ell=1}^{q}(x + w_\ell)$, note that P is coprime with $(X + w)$ as $w \notin \{w_\ell\}_{\ell=1}^{q}$. Algorithm $\mathcal{B}$ then returns the tuple $\left(w, P, \sigma_1'^{t'_{i^*}}/\sigma^{t_{i^*}}, \sigma_2'/\sigma_2\right)$ and wins the q-MSDH-1 challenge. The rewinding lemma (Lemma 6) implies that adversary $\mathcal{A}$ outputs another forgery with probability at least $\left(\varepsilon^2 - \varepsilon/|\Theta|\right)$ by considering (for the application of the lemma) X as the inputs of $\mathcal{A}$ (including its random tape) up to its to query to $\mathcal{H}_1$ in the first run and its response, excluding t_{i^*}; Y and Y' as t_{i^*} and t'_{i^*} respectively; Z and Z' as the inputs given to $\mathcal{A}$ strictly after the query to $\mathcal{H}_1$ in the first and second run respectively; f as the function which outputs 1 if $\mathcal{A}$ outputs another forgery and 0 otherwise. If adversary $\mathcal{A}$ does not output another forgery, $\mathcal{B}$ returns $\perp$. It follows that algorithm $\mathcal{B}$ wins the q-MSDH-1 challenge with probability at least $\left(1 - 1/|\Theta|\right)\left(\varepsilon^2 - \varepsilon/|\Theta|\right)$. As $1/|\Theta|$ is negligible, if ε were non-negligible, $\mathcal{B}$ would win the q-MSDH-1 game with non-negligible probability: a contradiction. Such an adversary $\mathcal{A}$ can therefore not exist.

In the general case, i.e., for an adversary which makes $q_{\mathcal{H}_1} > 1$ queries to $\mathcal{H}_1$ and wins the weak forgery game with probability ε, there exists an adversary $\mathcal{A}'$ which runs $\mathcal{A}$ as a subroutine, makes only one query to $\mathcal{H}_1$, and wins the q-MSDH-1 challenge with probability at least $\varepsilon/q_{\mathcal{H}_1}$. Indeed, consider $\mathcal{A}'$ an algorithm which chooses uniformly at random one of the $q_{\mathcal{H}_1}$ queries to $\mathcal{H}_1$ and forwards it to the challenger, and replies to the rest of the $\mathcal{H}_1$-queries with by choosing uniformly random values itself. It also forwards to the challenger all the other type of queries that $\mathcal{A}$ makes. When $\mathcal{A}$ outputs a forgery together with a list of public keys, if those latter are the ones in the query $\mathcal{A}'$ chose to forward, $\mathcal{A}'$ submits the forgery, otherwise it returns $\perp$. Algorithm $\mathcal{A}'$ then wins the weak forgery game with probability at least $\varepsilon/q_{\mathcal{H}_1}$. Therefore, if there exists an efficient adversary $\mathcal{A}$ that wins the weak forgery game with probability at least ε by making q non-adaptive signing queries and $q_{\mathcal{H}_1}$ queries to oracle $\mathcal{H}_1$, there exists an algorithm $\mathcal{B}$ with running time essentially twice that of $\mathcal{A}$, which wins the q-MSDH-1 game with probability at least $\left(1 - 1/|\Theta|\right)\left(\varepsilon^2 - \varepsilon/|\Theta|\right)/q_{\mathcal{H}_1}$. Under the q-MSDH-1 assumption, ε must be negligible, and the PS single-message multi-signature is thus EUF-wCMA secure. □

Proof (of Lemma 4). The proof consists in showing that if there exists an efficient algorithm that can win the forgery game for the modified PS k-message multi-signature scheme, then, if the SDL assumption holds, there exists an efficient algorithm that can win the weak forgery game of the original single-message signature-scheme. Let $\mathcal{A}$ be an adversary that wins the forgery game for the modified PS k-message multi-signature scheme, for an integer $k \geq 1$, with a non-negligible probability ε. As a signing query implies a query to $\mathcal{H}_0$, adversary $\mathcal{A}$ makes at most $q_{\mathcal{H}_0}$ signing queries. For $\ell \in [q_{\mathcal{H}_0}]$, denote by $m_\ell = (m_{\ell,1}, \ldots, m_{\ell,k})$ the messages for which $\mathcal{A}$ makes a signing query. Under the SDL assumption, if i^* is the index of the target (honest) signer, then the message $m = (m_1, \ldots, m_k)$ for which $\mathcal{A}$ outputs a forgery is such that

$\sum_{j=1}^{k} y_{i^*,j} m_j + y_{i^*,k+1} m_{k+1} \neq \sum_{j=1}^{k} y_{i^*,j} m_{\ell,j} + y_{i^*,k+1} m_{\ell,k+1}$ for $\ell \in [q_{\mathcal{H}_0}]$ (if ℓ if strictly greater than the number of signing queries, set $m_\ell = \perp$).

Indeed, would it not be the case, consider a algorithm $\mathcal{B}$ which runs $\mathcal{A}$ as a subroutine and interacts with an SDL challenger which outputs an instance $\left(g, \tilde{g}, Y = g^y, \tilde{Y} = \tilde{g}^y\right)$. Given a target (honest) signer index $i^* \in [n]$ from $\mathcal{A}$, algorithm $\mathcal{B}$ generates $x_{i^*} \in_R \mathbb{Z}_p$ and $a_j, b_j \in_R \mathbb{Z}_p$ for $j = 1, \ldots, k+1$, and sets and sends $\tilde{X}_{i^*} = \tilde{g}^{x_{i^*}}$, $\tilde{Y}_{i^*,j} = \tilde{g}^{a_j} \tilde{Y}^{b_j}$ for $j = 1, \ldots, k+1$ to $\mathcal{A}$. It implicitly sets $y_{i^*,j} = a_j + y b_j$.

To answer $\mathcal{H}_1$ queries, $\mathcal{B}$ chooses uniformly random values. To answer the ℓ-th $\mathcal{H}_0$ query on a new message m_ℓ, algorithm $\mathcal{B}$ prepares and stores a signature on m_ℓ, i.e., generates $m'_\ell, t_\ell \in_R \mathbb{Z}_p$, sets $\sigma_{\ell,1} \leftarrow g^{t_\ell}$ and

$$\begin{aligned}\sigma_{\ell,2} &\leftarrow \left(g^{x_{i^*} + \sum_{j=1}^{k} a_j m_{\ell,j} + a_{k+1} m'_\ell} \cdot Y^{\sum_{j=1}^{k} y_j m_{\ell,j} + y_{k+1} m'_\ell}\right)^{t_\ell} \\ &= \sigma_{\ell,1}^{x_{i^*} + \sum_j y_{i^*,j} m_{\ell,j} + y_{\ell,k+1} m'_\ell}.\end{aligned}$$

Algorithm $\mathcal{B}$ then replies with $\sigma_{\ell,1}$. Later, if $\mathcal{A}$ makes a signing query on m_ℓ, algorithm $\mathcal{B}$ replies with $(m'_\ell, \sigma_{\ell,1}, \sigma_{\ell,2})$. If $\mathcal{A}$ makes a signing query on a message m for which it has not made a $\mathcal{H}_0$-query yet, algorithm $\mathcal{B}$ proceeds as before but also outputs the signature instead of only storing it. If $\mathcal{A}$ makes a $\mathcal{H}_0$-query for a message for which it has already made a signing or $\mathcal{H}_0$ query, algorithm $\mathcal{B}$ answers as it priorly did.

When $\mathcal{A}$ eventually outputs a forgery (m', σ_1, σ_2) on a message m such that $\sum_{j=1}^{k} y_{i^*,j} m_j + y_{i^*,k+1} m_{k+1} = \sum_{j=1}^{k} y_{i^*,j} m_{\ell,j} + y_{i^*,k+1} m_{\ell,k+1} \mod p$ for some $\ell \in [q_{\mathcal{H}_0}]$ and a message m_ℓ for which it has made a signing query, then

$$\tilde{g}^{\sum_{j=1}^{k} a_j (m_j - m_{\ell,j}) + a_{k+1}\left(m' - m'_{\ell,j}\right)} = \tilde{Y}^{\sum_{j=1}^{k} b_j (m_j - m_{\ell,j}) + b_{k+1}\left(m' - m'_{\ell,j}\right)}.$$

Since the distribution of the b_j values conditioned on the input of adversary $\mathcal{A}$ is uniformly random (because of the a_j values), $\sum_{j=1}^{k} b_j (m_j - m_{\ell,j}) + b_{k+1} (m' - m'_{\ell,j}\Big) = 0 \mod p$ with probability $1/p$. Consequently, algorithm $\mathcal{B}$ wins the SDL challenge with probability at least $(1-1/p)\varepsilon$, which is non-negligible and the SDL assumption is thus contradicted. It follows that if the SDL assumption holds, then the message $m = (m_1, \ldots, m_k)$ for which $\mathcal{A}$ outputs a forgery is necessarily such that $\sum_{j=1}^{k} y_{i^*,j} m_j + y_{i^*,k+1} m_{k+1} \neq \sum_{j=1}^{k} y_{i^*,j} m_{\ell,j} + y_{i^*,k+1} m_{\ell,k+1}$ for $\ell \in [q_{\mathcal{H}_0}]$.

Consider then an algorithm $\mathcal{B}$ which runs $\mathcal{A}$ as a sub-routine and interacts with a challenger for the weak unforgeability game of the PS k-message multi-signature scheme. On the input of public parameters, $\mathcal{B}$ forwards them to $\mathcal{A}$, receives a target signer index i^*, and forwards it to the challenger together with $q_{\mathcal{H}_0}$ messages $w_1, \ldots, w_{q_{\mathcal{H}_0}}$ chosen uniformly at random. The challenger outputs a verification key $\left(\tilde{X}_{i^*}, \tilde{Y}_{i^*}\right)$ for signer i^* and signatures $\sigma_1, \ldots, \sigma_{q_{\mathcal{H}_0}}$.

Next, algorithm $\mathcal{B}$ generates $u_j \in_R \mathbb{Z}_p$ for $j = 1, \ldots, k+1$, and computes $\tilde{Y}_{i^*,1} \leftarrow \tilde{Y}$ and $\tilde{Y}_{i^*,j} \leftarrow \tilde{Y}^{u_j}$ for $j = 2, \ldots, k+1$. Algorithm $\mathcal{B}$ then out-

puts $vk_{i^*} \leftarrow \left(\tilde{X}_{i^*}, \tilde{Y}_{i^*,1}, \ldots, \tilde{Y}_{i^*,k+1}\right)$. This implicitly sets $sk_{i^*} = (x_{i^*}, y_{i^*,1} = y_{i^*}, y_{i^*,2} = u_2 y_{i^*}, \ldots, y_{i^*,k+1} = u_{k+1} y_{i^*})$. To answer $\mathcal{H}_1$ queries, $\mathcal{B}$ chooses uniformly random values. To answer the ℓ-th $\mathcal{H}_0$ query on a new message m_ℓ, algorithm $\mathcal{B}$ prepares and stores a signature on m_ℓ by setting

$$m'_\ell \leftarrow u_{k+1}^{-1} \left(w_\ell - \sum_{j=1}^{k} u_j m_{\ell,j} \right) \mod p$$

with $u_1 = 1$. Since $y_{i^*,1} w_\ell = \sum_{j=1}^{k} y_{i^*,1} u_j m_{\ell,j} + y_{i^*,1} u_{k+1} m'_\ell = \sum_{j=1}^{k} y_j m_{\ell,j} + y_{k+1} m'_\ell$, the tuple $(m'_\ell, \sigma_{\ell,1}, \sigma_{\ell,2})$ is a valid signature on m_ℓ. Algorithm $\mathcal{B}$ then replies the $\mathcal{H}_0$-query with $\sigma_{\ell,1}$. Later, if $\mathcal{A}$ makes a signing query on m_ℓ, algorithm $\mathcal{B}$ replies with $(m'_\ell, \sigma_{\ell,1}, \sigma_{\ell,2})$. If $\mathcal{A}$ makes a signing query on a message m for which it has not made a $\mathcal{H}_0$-query yet, algorithm $\mathcal{B}$ proceeds as before but also outputs the signature instead of only storing it. If $\mathcal{A}$ makes a $\mathcal{H}_0$-query for a message for which it has already made a signing or $\mathcal{H}_0$ query, algorithm $\mathcal{B}$ answers as it priorly did.

When $\mathcal{A}$ eventually outputs a list of verifications keys for $i \neq i^*$ and a forgery (m', σ_1, σ_2) on a new message m (i.e., for which no signing query was made) such that $\sum_{j=1}^{k} y_{i^*,j} m_j + y_{i^*,k+1} m_{k+1} \neq \sum_{j=1}^{k} y_{i^*,j} m_{\ell,j} + y_{i^*,k+1} m_{\ell,k+1} \mod p$ for all $\ell \in [q_{\mathcal{H}_0}]$, setting $w = \sum_{j=1}^{k} u_j m_j + u_{k+1} m'$, note that $y_{i^*,1} w \neq y_{i^*,1} w_\ell$ for $\ell \in [q_{\mathcal{H}_0}]$. Therefore, (σ_1, σ_2) is a valid forgery for the new message m. Algorithm $\mathcal{B}$ then perfectly simulates to $\mathcal{A}$ the challenger of the forgery game for the modified PS k-message multi-signature scheme, and wins with the same probability with which $\mathcal{A}$ wins the weak forgery game for the PS single-message multi-signature scheme. Hence the statement of the lemma. □

F Our Distributed Group Signatures with Threshold Opening

This section formally describes the key-generation and key-aggregation algorithms of the issuers and the issuance protocol for the construction in Section 5.2.

$\mathsf{IKG}(pp, i \in [n]) \rightarrow (ipk_i, isk_i, st_i))$: generate $(vk, sk) \leftarrow \mathsf{PSM.KG}(pp)$ and set $(pk_i, sk_i) \leftarrow (vk, sk)$. Denote by $\mathcal{R}_{\mathsf{PSM}}$ the relation of honestly generated PS multi-signature public and secret keys. Compute $\pi_i \leftarrow \mathsf{NIZK.Prove}\{sk_i : (pk_i, sk_i) \in \mathcal{R}_{\mathsf{PSM}}\}$. Set $ipk_i \leftarrow (i, pk_i, \pi_i)$ and $isk_i \leftarrow (ipk_i, sk_i)$. Initialize an empty state st_i. Return (ipk_i, isk_i, st_i). The vector of all issuer public keys is denoted $\boldsymbol{ipk}$.

$\mathsf{IKAggreg}(ipk_1, \ldots, ipk_n) \rightarrow ipk$: for $i \in [n]$, parse ipk_i as (i, pk_i, π_i). Verify that for all $i \in [n]$, $\mathsf{NIZK.Verf}(\tilde{g}, pk_i, \pi_i) = 1$. Compute an aggregated key $avk \leftarrow \mathsf{PSM.KAggreg}(pk_1, \ldots, pk_n)$. Set an return $ipk \leftarrow avk$.

The group public key gpk is set to $(\boldsymbol{ipk}, \boldsymbol{opk})$.

GJoin The protocol assumes a secure channel between $\mathcal{U}$ and every issuer $\mathcal{I}_i$ as well as a broadcast channel. Formally,

1. GJoin.U, on input (id, gpk),
   - choose $sk_{id} \in_R \mathbb{Z}_p^*$
   - $(a', h) \leftarrow \mathcal{H}_0(id)$
   - $h_{sk} \leftarrow h^{sk_{id}}$; $g_{sk} \leftarrow g^{sk_{id}}$
   - $\pi \leftarrow \mathsf{NIZK.Prove}\{sk_{id} \colon h_{sk} = h^{sk_{id}} \wedge g_{sk} = g^{sk_{id}}\}$
   - generate $p_1, \ldots, p_{t_O} \in_R \mathbb{Z}_p$ and set $P \leftarrow sk_{id} + \sum_{\ell=1}^{t_O} p_\ell X^\ell \in \mathbb{Z}_p[X]$
   - for $i \in [n_O]$, compute $s_i \leftarrow P(i)$
   - for $\ell \in [t_O]$, compute $h_\ell \leftarrow h^{p_\ell}$
   - for all $i \in [n_O]$:
     - $r_i \leftarrow_\$ \mathbb{Z}_p$
     - $\tilde{C}_i \coloneqq (\tilde{C}_{i,0}, \tilde{C}_{i,1}) \leftarrow \left(\tilde{g}^{r_i}, \tilde{f}_i^{r_i} \tilde{Y}_0^{s_i}\right)$
     - $\pi_i \leftarrow \mathsf{NIZK.Prove}\left\{r_i \colon \tilde{C}_{i,0} = \tilde{g}^{r_i}, e\left(h, \tilde{C}_{i,1}/\tilde{f}_i^{r_i}\right) = e\left(h_{sk} \prod_{\ell=1}^{t_O} h_\ell^{i^\ell}, \tilde{Y}_0\right)\right\}$
   - set $L[id] \leftarrow \left(g_{sk}, h_{sk}, h_1, \ldots, h_{t_O}, \pi, \left(\tilde{C}_i, \pi_i\right)_{i \in [n_O]}\right)$
   - broadcast written to all $\mathcal{I}_i$

2. GJoin.I, for $i \in [n_I]$, on input $(st_i, isk_i = (i, x_i, y_{0,i}, y_{1,i}), id, I, gpk)$
   - abort if $id \in st_i$
   - upon receiving written from $\mathcal{U}$:
     - $(a', h) \leftarrow \mathcal{H}_0(id)$
     - parse $L[id]$ as $\left(g_{sk}, h_{sk}, h_1, \ldots, h_{t_O}, \pi, \left(\tilde{C}_i, \pi_i\right)_{i \in [n_O]}\right)$
     - $\mathsf{NIZK.Verf}(g, h, g_{sk}, h_{sk}, \pi) \stackrel{?}{=} 1$
     - for $j \in [n_O]$, $\mathsf{NIZK.Verf}\left(h, (h_\ell)_{\ell=1}^{t_O}, \tilde{Y}_0, \tilde{f}_j, \tilde{C}_j, \pi_j\right) \stackrel{?}{=} 1$
     - $\Sigma_{i,2} \leftarrow h^{x_i + y_{i,1} a'} h_{sk}^{y_{i,0}}$
     - $st_i \leftarrow st_i \cup \{id\}$
     - send $\Sigma_{i,2}$ to $\mathcal{U}$ over a secure channel

3. GJoin.U, upon receiving $\Sigma_{i,2}$ from all $\mathcal{I}_i$,
   - $ipk \leftarrow \mathsf{PSM.KAggreg}(\boldsymbol{pk})$
   - $\Sigma \leftarrow \mathsf{PSM.SAggreg}(\boldsymbol{pk}, sk_{id}, (a', h, \Sigma_{i,2})_{i=1}^{n_I})$
   - $\mathsf{PSM.Verf}(ipk, sk_{id}, \Sigma) \stackrel{?}{=} 1$
   - return $\mathbf{gsk}[id] \leftarrow (sk_{id}, \Sigma)$.