Practical Semi-Open Chat Groups for Secure Messaging Applications

Alex Davidson
LASIGE, Universidade de Lisboa

Luiza Soezima
Université Grenoble Alpes

Fernando Virdia
King's College London

***Abstract*—Chat groups in secure messaging applications such as Signal, Telegram, and Whatsapp are nowadays used for rapid and widespread dissemination of information to large groups of people. This is common even in sensitive contexts, associated with the organisation of protests, activist groups, and internal company dialogues. Manual administration of who has access to such groups quickly becomes infeasible, in the presence of hundreds or thousands of members.**

We construct a practical, privacy-preserving reputation system, that automates the approval of new group members based on their reputation amongst the existing membership. We demonstrate security against malicious adversaries in a single-server model, with no further trust assumptions required. Furthermore, our protocol supports arbitrary reputation calculations while almost all group members are offline (as is likely). In addition, we demonstrate the practicality of the approach via an open-source implementation. For groups of size 50 (resp. 200), an admission process on a user that received 40 (resp. 80) scores requires 1312.2 KiB (resp. 5239.4 KiB) of communication, and 3.3 s (resp. 16.3 s) of overall computation on a single core. While our protocol design matches existing secure messaging applications, we believe it can have value in distributed reputation computation beyond this problem setting.

1. Introduction

The *point-to-point* communication model of the Diffie-Hellman era is rapidly losing relevance relative to modern wide-scale message dissemination models. Today, secure messaging groups provide *town squares* for wide-scale organising and information dispersal. Generally, messaging services allow group chats to be formed by direct invite by one or more administrators (*closed groups*), or by generating and openly sharing an invite link that automatically grants access to anybody that receives it (*open groups*). However, as groups grow into hundreds, or even thousands of members, manual administration of who has access to potentially sensitive content becomes infeasible to maintain — even when such responsibility is split across multiple individuals. In protest/active organisation [1] and internal company dialogues, the management of such access is pivotal to ensuring that outsiders/competitors cannot take advantage of learning sensitive information.

When deciding whether to give access to a secure channel to a new party, an alternative to manual access control is to automatically poll already-trusted parties, to compute a *reputation* score for the potential new member. In such scenarios, we assume that anyone who has a positive reputation score with current group members is likely someone who would be admitted under a manual administration framework. Clearly, canvassing such opinions directly exposes the voters to leaking their (preferential) social network. Therefore, such reputation systems must provide meaningful privacy guarantees for individual scores.

General-purpose multi-party computation [32] (MPC) or e-voting [7] protocols appear as potential solutions. However, current incarnations of such protocols are at best cumbersome, and at worst completely unviable. For instance, in many real-world systems voters are frequently offline and under strict bandwidth and power constraints. Furthermore, MPC and e-voting designs often rely on strong trust assumptions over the non-collusion of protocol entities, which cannot be meaningfully attained in real-world deployments (where such entities have explicit co-dependencies). Even in more targeted reputation systems, such trust assumptions remain present (e.g. in the form of utilising mixnets of non-colluding nodes) [18].

Our work. We formalise, construct, and implement reputable group formation systems for secure messaging applications, that allow for practical *semi-open* groups. A semi-open group allows joining via URLs, but admission is managed automatically, rather than manually. The admission process depends on the joining user's perceived *reputation* within the group, based on past votes issued by group members. At all times, it is guaranteed that the confidentiality of votes (either counted or not counted) is maintained, as well as the anonymity of the voters themselves. The protocol takes place between a single online device in the group, the application server (e.g. run by Signal), and the target user. After completion, the admitted user joins the standard continuous group-key agreement protocol used by contemporary messaging applications [28].

In terms of the design, we define a new protocol (Section 6) that relies on minimal cryptography, based only on standard cryptographic groups (e.g. Ristretto255 [13]). Along the way, we build novel shuffled exponentiation interactions (Section 3) that may be of independent interest. With these building blocks, we characterise system (Section 4) and security models (Section 5) that such protocols must satisfy. Ultimately, we show that our construction maintains security against a malicious

adversary that corrupts multiple group-affiliated individuals in parallel. We implement our construction (Section 7) and show that it can handle groups of hundreds of members with low and linearly scaling communication and computation overheads (e.g. 5239.4 KiB, and 16.3 s of overall computation for a group of 200). Note that performance is completely *independent* of the size of the universe of users,[1] and only depends on the size of the group admitting the U_{i^*}. We discuss remaining limitations and avenues for future work in Section 9, and note that our protocol is amenable to adaptations that guarantee more advanced functionality and security properties, in the face of stronger collusion models (Section 8).

Formal contributions.

- A novel and practical verifiable permuted exponentiation (VEP) protocol, based on the hardness of Decisional Diffie-Hellman (DDH) problem. Such a protocol takes a list $A = (a_1, \ldots, a_n) \in \mathbb{G}^n$ of group elements, and produces a list $B = (a^k_{\sigma(1)}, \ldots, a^k_{\sigma(n)})$ for some secret exponent $k \in \mathbb{F}_p$, and secret permutation $\sigma : [n] \mapsto [n]$.
- An ideal system and security model for constructing practical reputation tallying protocols, and proving security against malicious adversaries.
- A practical construction of such a tallying system, based on DDH, said VEP interaction and NIZK proofs of discrete log relations. We also provide adaptations that handle more complex collusion scenarios, such as those involving the server and any user, and additional functionality requirements.
- An open-source implementation and benchmarks[2] demonstrating the real-world viability of our system.

1.1. Technical Overview

We now introduce the reader to the technical heart of our work. During design, our starting point was functionality. Ideally, we wanted a system where the reputation of any user U_{i^*} could be computed by a group G of individuals. While this could be done using a generic e-voting scheme, such a solution would require the group members being online during the voting phase for their vote to be counted. It would also mean that any individual $\mathsf{U}_i \in \mathsf{G}$ may have to vote multiple times for the same user, if U_{i^*} tried to access multiple groups where U_i was a member of.

Our approach permits any individual U_i to vote on a target U_{i^*} at any point during their lifetime, even before group formation. These votes are encrypted into ballots, that are delivered to the server, and collected under U_{i^*}'s identity. Our core contribution is building intersection functionality that allows, at a later stage, to fetch ballots under U_{i^*} identity, detect which belong to members of a given group G that U_{i^*} is trying to access, and decrypt them, without linking any ballot to a specific group member. Notably, we do not discuss a specific measure of reputation. Rather, we view our protocol to be essentially a very specific multi-party computation (MPC) scheme, that instantiates this "intersection" functionality, and returns the decrypted votes "belonging" to a group. This allows significant flexibility in the choice of reputation function to compute.

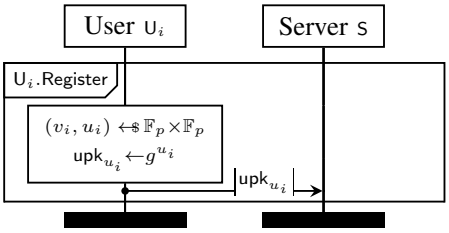

Figure 1. User registration phase.

To illustrate the overall approach, we will describe here a simpler variant of our scheme that is secure only in an honest-but-curious setting, that allows a single vote to be expressed by U_i on U_{i^*}, and that assumes an adversary corrupting the server does not corrupt any other entity associated (or who becomes associated) with G. Ultimately, we give a protocol that satisfies malicious security (Section 6), and we show that the other limitations can also be overcome, as described in Sections 8.1 and 8.2.

Overall design. Throughout the following protocol phases, g generates a group of prime-order p, where discrete log and decisional Diffie-Hellman (Definition 2.1) are hard problems. In the following description, we will describe the protocol steps, in conjunction with figs. 1 to 5. Each figure defines a set of algorithms that are later defined formally in Section 6.

The first phase of the protocol is user registration, see fig. 1. Here, a user U_i generates two secret exponents v_i, u_i, and a public group element $\mathsf{upk}_{u_i} = g^{u_i}$. The element upk_i acts as a public key in the system, allowing other users to vote on U_i, while v_i can be seen as a voting secret key, allowing U_i to vote on different users.

At any point in time after registration, user U_i may decide to express a vote on a target user U_j, see fig. 2. U_i may choose a score $x_{i,j}$ from a polynomial-sized domain D, and deterministically encrypt it in the exponent as a ballot $y_{i,j} = (\mathsf{upk}_{u_j})^{v_i} \cdot g^{x_{i,j}}$. This vote can then be sent via an anonymous channel (e.g. Tor [15] or Oblivious HTTP [31]) to the server S and U.[3] It should be noted that since g^{v_i} is never published, U_j cannot recover the vote from the ballot.

At any time, a small number of users can create a "semi-open" group G (see fig. 3). Here, enough users should be present that an automatic reputation value could be meaningfully estimated. In this initial phase, all users are considered trustworthy, and are automatically included. During group creation, the server generates a group-specific private exponent s_G and a corresponding public key $\mathsf{spk}_\mathsf{G} = g^{s_\mathsf{G}}$, which is provided to every group member (or possibly

1. As opposed to solutions based on ring signatures.
2. https://github.com/luizabrs/semi-open-messaging-groups

3. Note that protocol variants may choose to not send such ballots to the user themselves, see Section 8.2.

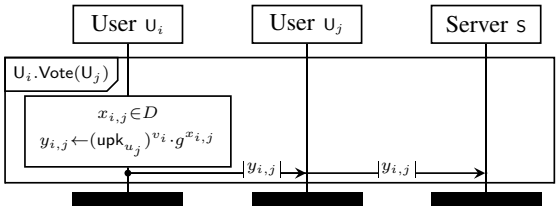

Figure 2. Voting process.

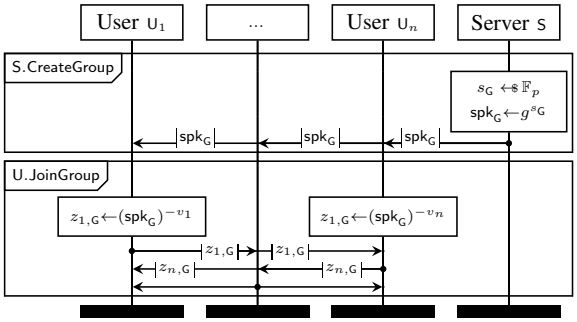

Figure 3. Initial group creation.

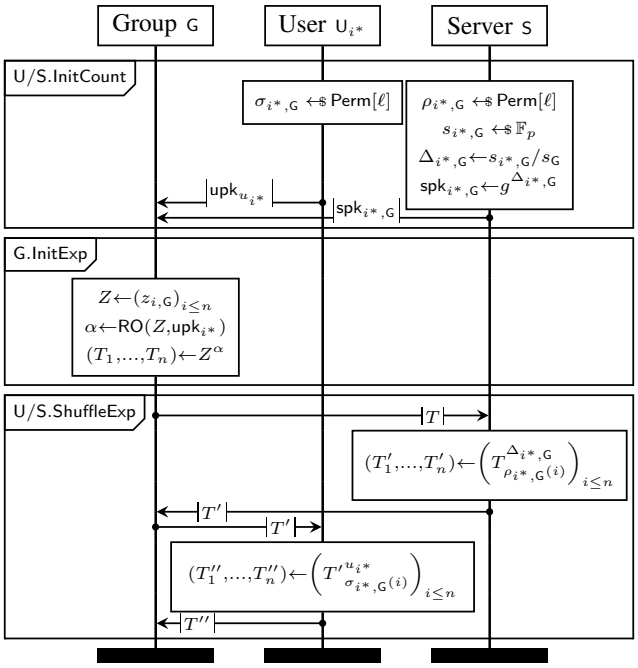

Figure 4. Join request, utilising verifiable permuted exponentiations.

to a specific "admin" member that created the group, and that distributes it to the other members). Every group member uses ther private exponent v_i to generate a group-specific "intersection tags" $z_{i,\mathsf{G}} = (\mathsf{spk}_\mathsf{G})^{-v_i}$, that they share with the other group members. These tags will allow ballots generated by U_i to be detected during reputation tallying of an external user, even if U_i is offline.

Finally, a user U_{i^*} requests to join the group G, see fig. 4. Here, we work under the assumption that the group members have a way of coordinating, so that all operations performed by them can be considered to happen under a "Group" identity. For example, this can be obtained by having an administrator user that is required to be online to process a join request (though this role can change hands arbitrarily). We assume all group members share the view of the "Group" identity.

Our main objective is to be able to intersect the set of ballots received by U_{i^*}, $W_{i^*} = \{g^{u_{i^*} v_j + x_{j,i^*}}\}_j$, with the intersection tags $z_{i,\mathsf{G}} = (\mathsf{spk}_\mathsf{G})^{-v_i}$. Mathematically, this would be possible by getting the u_{i^*} exponent on the tags, and the s_G exponent on the ballots, and then checking for whether the resulting product falls in the set $\{g^x \mid x \in D\}$.

In practice, a few extra steps are required to achieve the unlinkability of tags and ballots to group members within the join request, and across different join requests. In particular, this is achieved by having the tags be randomly shuffled before intersection with the ballots, via a novel verifiable permuted exponentiation protocol (Section 3). Furthermore, we have the server use an ephemeral group exponent specific to the "join" request by U_{i^*}. After picking random shuffles $\rho_{i^*,\mathsf{G}}$, $\sigma_{i^*,\mathsf{G}}$ and ephemeral exponent $s_{i^*,\mathsf{G}}$ (and corresponding public key $\overline{\mathsf{spk}_{i^*,\mathsf{G}}}$), the group sends the list of intersection tags, obfuscated with an exponent α. Server and external user will then shuffle and add their exponents to the tags.

Finally, the server will provide a copy of the ballots received by user U_{i^*}, with the ephemeral server exponent added. The group can then intersect the ballots and recover the votes, by computing a discrete logarithm of the $g^{s_{i^*,\mathsf{G}} \cdot x_{j,i^*}}$ with respect to $\overline{\mathsf{spk}_{i^*,\mathsf{G}}}$, which is possible in polynomial $O(|D|)$ time due to the polynomial-sized vote domain. The plaintext votes are then used to determine whether the external user should be automatically admitted to the group. If that's the case, the newly admitted user U_{i^*} will provide their intersection tag and receive the ones from the other group members.

1.2. Wider Discussion and Limitations

Design choices. We observe a few interesting properties of this design. Firstly, it allows the reputation score of any user to be relative to the group computing it. Differently than for schemes like AnonRep [33], where reputation is global, this provides the benefit of better matching our intuition of reputation (a person can have good and bad reputation, depending on whom one asks). This also protects from trivial *sybil* attacks, such as creating thousands of "fake" accounts to negatively vote on someone: even if this happened, unless the sybil accounts were in the group, their votes would cause no effect on the user's reputation.

The use of secret shuffles to provide anonymity, and the requirement of matching ballots with the group also remove the need for ring or group signature schemes, otherwise popular with reputation systems. While such

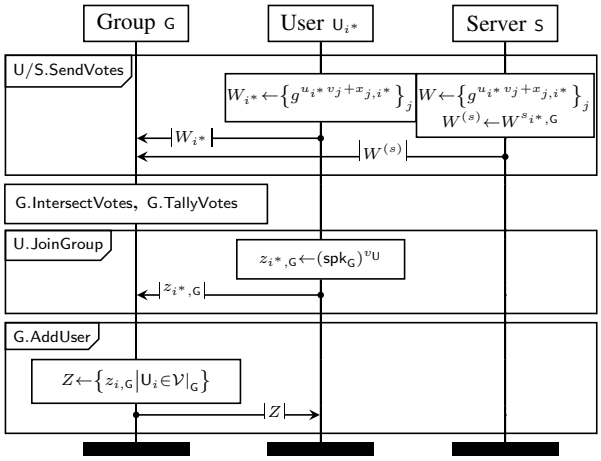

Figure 5. Ballot intersection and tally.

schemes provide a good interface, they require the signing party to have access to all the public keys in the group. As we aim to be able to vote before group creation, as to reduce the burden on the voting party, this would mean holding every public key in the messaging system. With Signal, WhatsApp, and related services having hundreds of millions of active users, this would not be a realistic option.

Intrinsic limitations. Our design goal is that of producing a "practical" design. In our view, practicality requires the ability of voting once and having the resulting ballot count in multiple intersections and allowing ballot intersections while group members are being offline. Group members should not be required to come online to manually vote on an external user when someone wants to join the semi-open group. The base protocol in Section 6 satisfies these constraints in the presence of malicious adversaries, by having parties to prove they performed their steps correctly, and having them check that the other parties also performed their steps correctly. These practicality constraints imply some limitations of the resulting protocol.

First, allowing intersections while (some) group members are offline means that these users cannott check in real time the correctness of the intersection performed (from their point of view), meaning that their ballots may not be included during it. Instead, since all parties publish a transcript of the intersection, users can check *a posteriori* that the intersection was performed correctly, and will blow the whistle on the malicious party otherwise. Of course, in the meantime an external user that should not have been admitted, may have been admitted to the group, and compromised some of the communication. This seems to be an inherent trade-off caused by the lower strictness of the admission procedure. Something similar may happen with the reputation computation: it could be that the intersection is performed correctly, but the function used to compute a final "score" for the external user is not adequate to determining security. This is a similar intrinsic risk in automating group admission.

Second, there is also the possibility that an external user to the group has not accumulated enough ballots on themselves, and a score cannot meaningfully be computed because the intersection of ballots is empty. Again, this is an intrinsic limitation to automating a reputation computation. We believe the safe approach here would be to fail-close: notify the group admin(s) that an external user has requested to join but does not have a computable reputation, and delegate to them the decision of admitting them or not, akin to the closed group setting. The problem of having no history as a user is intrinsic to any reputation system, not just our design. Yet, we still believe semi-open groups may help reducing administrator burden within tight-knit communities, such as universities or work environments, where people do connect.

Lastly, there is the issue of the multifaceted nature of people's identities. Alice may think that Bob is a decent person and should be allowed automatically into their workplace's social events. However, she may disagree with his political views, and would rather not vote to have him automatically admitted into activism chat groups she belongs to. How should she vote? A technological solution could be that of having Alice (and any other user) hold different identities, say Alice_w for work, and Alice_p for politics. She could then vote twice on Bob, with a high score under Alice_w and a low one under Alice_p. When joining a group she could hand over intersection tags from either one of her identities, depending on the nature of the group (or provide a random group element, if she does not want to hand over any tags useful to intersecting her ballots). This would allow her to express a more fine-grained opinion on Bob as a person. However, this may be overkill for most users in practice: we suspect that messaging groups on non-sensitive topics may not require a semi-open policy. Instead users may be happy holding just one identity regarding some sensitive topic of their interest, such as politics, and hand over random group elements as intersection tags otherwise.

1.3. Related Work

The problem of ascertaining reputation within localised communities has received significant attention.

Cryptographic reputation systems. Recent developments in so-called *privacy-preserving reputation systems* provide the closest approximation to the required functionality. Referring to the excellent survey of [18, Table 1], we see a dearth of existing protocols that provide strong privacy guarantees. In particular, the stand-out options appear to be AnonRep [33], ARM4FS [30], and pRate [26]. In the case of ARM4FS a voting system for file sharing use-cases, a trusted third-party TTP is used to faciliate voting, and all ballots are linkable to voters by said TTP. In AnonRep, linkable ring signatures [27] are used to cast single votes in forum setting. Here, a mixnet (of non-colluding parties) is used to verifiably shuffle said votes, so that they can be used to perform elections without linking back to voter identities. Security guarantees for AnonRep

are not proven, and the linkable ring signature approach requires knowing every public key in the sytsem. pRate is configured to providing trust in a system where ratings are produced on various entities, who can later choose to engage in dialogue. pRate utilises a TTP to handle the ballots produced by each entity — voter anonymity is not provided, but confidentiality is maintained — and the construction utilises pairing-based cryptography and various non-standard zero-knowledge proofs. In all cases, the presence of a TTP, or non-colluding parties, makes such systems hard to deploy. The more recent work of PRSONA [19] builds on top of similar ideas to AnonRep, but requires tight-knit communities, and the security of the system still relies on non-collusion assumptions, and the cryptographic properties of the system are never proven. Finally, Bell and Eskandarian [4] recently demonstrated an anonymous complaint aggregation system, that allows for tallying scores produced by multiple users in an anonymous way. As in previous works, this requires at least two non-colluding servers to distribute the opening and verification of reputation scores.

Webs-of-Trust. Adjacent to cryptography, but not providing formal cryptographic guarantees, two popular solutions are using Certificate Authorities (CA) [29] and Webs-of-Trust (WoT) [25]. The first is the most popular solution used in conjunction with authenticated key-exchange (usually TLS), and relies on CAs, essentially trusted third-parties that attest for the reputation of users within a Public Key Infrastructure (PKI). Someone retrieving a public key would check for a signature by a CA as a signifier of validity of the key. The WoT model replaces the hierarchical model of CAs, with a distributed horizontal one, where users within the PKI all act as CA on each other, by only signing public keys they consider reputable. Someone retrieving a public key would check for signatures by other holders of public keys that they trust, using the number and origin of the signatures as a signifier of validity of the key.

Trust systems. The problem of reputation within more general dynamic systems of entities has also been treated within the literature on trust systems, where the issue of messaging group formation finds a parallel in the literature on Virtual Organizations (VO). Here, security properties are often derived from economic incentives [24], albeit cryptographic designs have also been proposed [23].

2. Notation and Preliminaries

Sets and permutations. We write $[n] = \{1, \ldots, n\}$, and denote $\mathsf{Perm}[n]$ as the set of all permutations defined over $[n]$. We denote by Id the permutation such that $\mathsf{Id}([n]) = [n]$. For a set X, we write $X' \leftarrow \mathsf{Shuffle}(X)$ to denote a randomly permuted version of X. For a probability distribution D over a set S, we write $Z \leftarrow D^{\ell}$ to denote the sampling of δ elements from S, to create the set Z.

Algorithms. We use PPT to refer to algorithms that terminate with a result in probabilistic polynomial-time.

2.1. Cryptographic Groups

We make use of cyclic groups $\mathbb{G} = \mathbb{G}(\lambda)$, where the order of the group $q = \mathsf{poly}(\lambda)$ is prime (using multiplicative notation). The scalar field associated with $\mathbb{G}$ will be written as $\mathbb{F}(\mathbb{G})$ (sometimes simply as $\mathbb{F}$ when the context is obvious), where the order of $\mathbb{F}$ is defined to be $p = \mathsf{poly}(\lambda)$. We assume that the groups we use admit hard instances of the discrete log and decisional Diffie-Hellman (Definition 2.1 below) problems.

Definition 2.1 (Decisional Diffie-Hellman)**.** *The Decisional Diffie-Hellman (DDH) problem is defined with respect to a PPT algorithm $\mathcal{B}$ and a group $\mathbb{G}$. $\mathcal{B}$ is tasked with distinguishing the distributions (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, r), where g is a fixed generator of $\mathbb{G}$, $a, b \leftarrow\!\!\$\, \mathbb{F}(\mathbb{G})$, and $r \leftarrow\!\!\$\, \mathbb{G}$.*

It is widely assumed that the DDH problem is hard to solve in practical instantiations of groups. In other words, $\mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{B}}(\lambda) < \mathsf{negl}(\lambda)$ for all PPT algorithms $\mathcal{B}$.

2.2. NIZK Proof Systems

We recall standard definitions about non-interactive proof systems.

Definition 2.2 (Non-interactive proof system)**.** *Let $\mathcal{R} \subset \mathcal{X} \times \mathcal{Y}$ be a relation with witness space $\mathcal{X}$ and statement space $\mathcal{Y}$. We say that $\Pi = (\mathsf{Prove}, \mathsf{Verify})$ is a non-interactive proof system for $\mathcal{R}$ with proof space $\mathcal{PS}$ if:*

- Prove *is an efficient probabilistic algorithm that on input a witness-statement pair $(x, y) \in \mathcal{R}$, returns a proof $\pi \leftarrow \mathsf{Prove}(x, y)$, $\pi \in \mathcal{PS}$,*
- Verify *is an efficient deterministic algorithm that, invoked on a statement y and a proof π, returns a bit $b \leftarrow \mathsf{Verify}(y, \pi)$, where $b = 1$ denotes acceptance and $b = 0$ rejection. If $b = 1$, then π is a valid proof for y.*

Definition 2.3 (Completeness)**.** *Let $\Pi = (\mathsf{Prove}, \mathsf{Verify})$ be a non-interactive proof system for some relation $\mathcal{R} \subset \mathcal{X} \times \mathcal{Y}$. We say Π is complete, if for any $(x, y) \in \mathcal{R}$, $\Pr[\mathsf{Verify}(y, \mathsf{Prove}(x, y)) = 1] = 1$.*

Definition 2.4 (Non-interactive knowledge soundness)**.** *Let $\Pi = (\mathsf{Prove}, \mathsf{Verify})$ be a non-interactive proof system for some relation $\mathcal{R} \subset \mathcal{X} \times \mathcal{Y}$, with proof space $\mathcal{PS}$. Let $L_{\mathcal{R}}$ be the language of true statements of $\mathcal{R}$. Then knowledge soundness specifies that, for any statement $(x, y) \notin \mathcal{R}$, an adversary $\mathcal{A}$ cannot produce a valid proof except for negligible probability, or $\mathsf{Adv}^{\mathrm{nisnd}}_{\Pi,\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.*

Knowledge soundness implies equivalently that there is a PPT algorithm $\mathsf{Extract}$, such that:

$$\Pr\left[z \neq x \;\middle|\; \begin{matrix} \pi^* \leftarrow \mathsf{Prove}(x,y) \\ 1 \leftarrow \mathsf{Verify}(y^*, \pi^*) \\ z \leftarrow \pi^*.\mathsf{Extract}(y^*) \end{matrix}\right] < \mathsf{negl}(\lambda).$$

In other words, $\mathsf{Extract}$ extracts the witness $x \in \mathcal{X}$, with all but negligible probability.

Definition 2.5 (Non-interactive zero knowledge)**.** *Let Π be a non-interactive proof system for some relation $\mathcal{R} \subset \mathcal{X} \times \mathcal{Y}$,*

$\mathsf{Prove}(z, (g, \boxed{a}), (Z := g^z, \boxed{A := a^z}))$	$\mathsf{Verify}((g, \boxed{a}), (Z, \boxed{A}), \pi)$
$t \leftarrow\$ \mathbb{F}(\mathbb{G})$	$(c, s) \leftarrow \pi$
$D \leftarrow g^t$	$D' \leftarrow g^s Z^{-c}$
$\boxed{E \leftarrow a^t}$	$\boxed{E' \leftarrow a^s A^{-c}}$
$c \leftarrow \mathsf{RO}(g, Z, D, \boxed{a, A, E})$	$c' \leftarrow \mathsf{RO}(g, Z, D', \boxed{a, A, E'})$
$s \leftarrow t + cz$	
return $\pi = (s, c)$	**return** $c \stackrel{?}{=} c'$

Figure 6. Basic description of Schnorr-like proofs for non-interactive DLog [16] and DLEQ [9] proof systems.

with proof space $\mathcal{PS}$. We say that Π provides non-interactive zero-knowledge if there exists an efficient simulator Sim *for Π, such that* Sim *can produce indistinguishable statements $y \in \mathcal{Y}$, even without knowledge of a valid witness $x \in \mathcal{X}$. We denote the advantage of an adversary attempting to distinguish such statements by $\mathsf{Adv}^{\mathsf{nizk}}_{\Pi,\mathcal{A},\mathsf{Sim}}(\lambda)$.*

2.3. Discrete-log Proofs

Proof of Knowledge of Discrete Log (DLog). We use standard Schnorr proof formulation for constructing NIZKs that prove knowledge of a discrete logarithm value, or so-called discrete log identification systems (denoted DLog), as described by Fiat and Shamir [16]. In other words, after computing $(z \in \mathbb{F}(\mathbb{G}), g^z \in \mathbb{G})$, an entity can prove knowledge of z without revealing it. For ensuring self-containment of this work, we give a formal construction of the proof and verification steps for DLog in fig. 6, where the fact that DLog satisfies completeness, soundness, and non-interactive zero-knowledge follows from [16].

Proof of Discrete Log Equivalence (DLEQ). We give a basic formulation of a DLEQ protocol in fig. 6 (including the additional steps in grey). This formulation is based on the simple formulation given by Chaum and Pedersen [8], that has been used in previous verifiable pseudorandom function protocols [11], [17], [22]. The fact that it satisfies completeness, soundness, and non-interactive zero-knowledge follows immediately from [9]. Note that, henceforth we abuse notation throughout, and write $\pi \leftarrow \mathsf{DLEQ.Prove}(z, Z, \{(a_i, A_i)\}_{i\in[\ell]})$ to be ℓ invocations of the standard single-input proof system shown, where π is then a set of ℓ individual DLEQ proofs, and $\pi[i] \leftarrow \mathsf{DLEQ.Prove}(z, Z, a_i, A_i)$. However, it is worth noting that such proofs can also be batched into a single element, in cases where shuffle-compatibility is not required [12], [21].

Shuffled DLEQ. In Section 3, we also require a non-interactive zero-knowledge proof of exponentiation-and-shuffle, so that given a key pair (g, h), and tuples $(g_1, \ldots, g_n)$ and $(h_1, \ldots, h_n)$, the prover can argue that it knows a secret exponent x and secret permutation σ, such that $h = g^x$, and $h_i = g^x_{\sigma(i)}$. To achieve this, we use the generic compiler of Haines and Müller [20] for turning a "shuffle-compatible" sigma protocol (SCSP) for a relation $\mathcal{R}$ into a (standard) sigma protocol for a "shuffled" combined relation $\mathcal{R}_{\mathsf{Shuffle}}$. We can then apply the compiler to a a shuffle-compatible version of the Chaum-Pedersen protocol [8] for proving discrete logarithm equivalence (DLEQ), and compile the resulting sigma protocol into a non-interactive proof system using the Fiat-Shamir transform [16]. Note that we can instantiate a standard DLEQ proof system (without shuffle) from this same formulation, by applying an identity permutation. However, for performance reasons, instantiating a standard DLEQ from the Schnorr variant shown in fig. 6 is usually more efficient. In the full version of this work, we formally spell out the shuffle-variant of DLEQ.

2.4. Malicious Security in Protocols

For a protocol Π, we consider participants $P_1, \ldots, P_n$ and an adversary $\mathcal{A}$ that can corrupt up to $l < n$ parties. We will assume corruptions are performed selectively, which means that $\mathcal{A}$ will choose which parties to corrupt before the protocol starts. Since we consider $\mathcal{A}$ to be malicious, any corruption of P_i will give $\mathcal{A}$ access to the entire internal state of P_i, and will allow $\mathcal{A}$ to pick the inputs of P_i during the protocol. Furthermore, $\mathcal{A}$ may deviate from the protocol specification arbitrarily: for example, by using malformed inputs, or triggering aborts arbitrarily. From a notational perspective, for an adversary that corrupts $l > 1$ parties, we will write $\mathcal{A} = (\mathcal{A}_{P_{i_1}}, \ldots, \mathcal{A}_{P_{i_l}})$ for $i_j \in [n]$, to refer to the individual algorithms that run for each of the participants.

For proving security of Π against all adversaries $\mathcal{A}$, we use the real/ideal-world paradigm for proving security in the standalone model. Therefore, let $\mathcal{F}$ be an ideal functionality that implements the core functionality of Π. Then we consider Π secure if the view of $\mathcal{A}$ in Π is computationally indistinguishable from a view constructed by some PPT simulator algorithm Sim, that only has access to $\mathcal{F}$.

Input extraction and step verification. In practice, in order for Sim to interact with $\mathcal{F}$, it must derive some input $x_{i_j} \leftarrow \mathcal{A}_{P_{i_j}}$ for each P_{i_j} that $\mathcal{A}$ corrupts. These inputs are then submitted to $\mathcal{F}$, and then Sim learns the output $y \leftarrow \mathcal{F}(x_{i_1}, \ldots, x_{i_l})$, where y is calculated using the implicit inputs of honest parties in the protocol. Since $\mathcal{A}$ may deviate arbitrarily, we must use tools that allow Sim extract such inputs in polynomial-time. In principle, our primary tools for doing this will be the usage of zero-knowledge proof systems that satisfy knowledge soundness, and the extractability of random oracles (that are managed by Sim). Such mechanisms will also be used to ensure that adversarial parties continue to abide by the protocol specification throughout.

Handling aborts. The Sim must abort the protocol whenever it receives a request from $\mathcal{A}$ to abort. As such, while we guarantee the confidentiality of the protocol on all executions, as well as correctness for full executions, we will assume that the adversary can simply terminate any protocol execution.

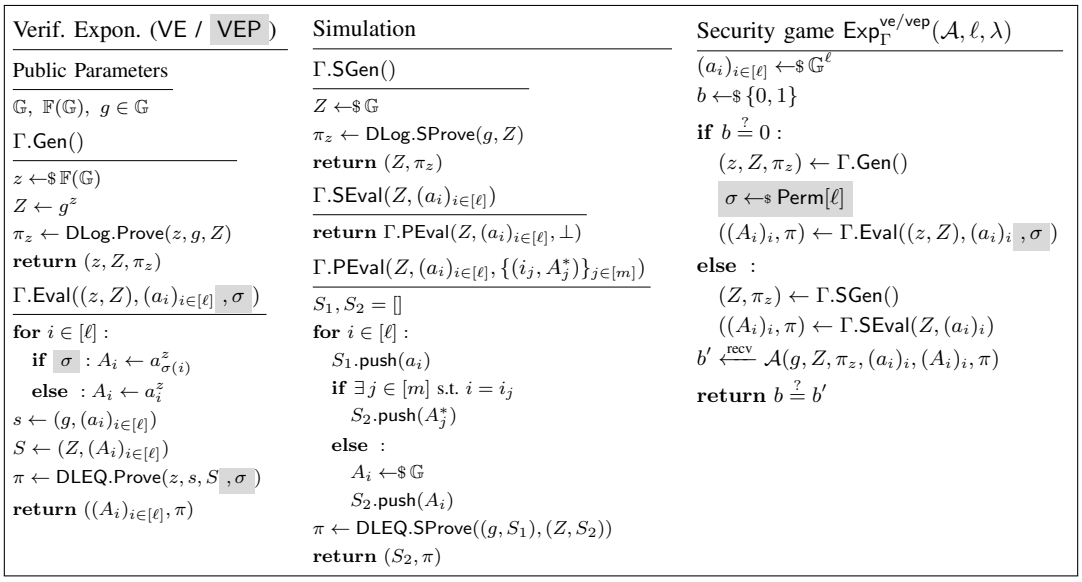

Figure 7. Verifiable exponentiation protocol and security model. Grey backgrounds denote steps used for instantiating the permuted variant, for $\sigma \neq \mathsf{Id}$.

3. Verifiable Permuted Exponentiation

In this section, we discuss concrete protocols for instantiating simple verifiable exponentiation protocols, that essentially perform exponentiations as-a-service that receivers can verify are correct with respect to previously committed public keys. In effect, such protocols consider a client that holds some group element $a \in \mathbb{G}$, a server that holds a scalar value $z \in \mathbb{F}(\mathbb{G})$, and a public generator $g \in \mathbb{G}$ known to both parties. The client eventually learns a^z, and is provided a proof that a^z has been computed correctly.

We give a formal construction of two such protocols in fig. 7. The first, VE, that follows this exact framework, and second, VEP, that allows the server to introduce a hidden shuffle/permutation when returning the elements, that the client cannot discern. Intuitively, the security property that we require is that the server-side exponentiation service can be simulated without knowledge of the private exponent or permutation. This leverages the non-interactive zero-knowledge properties of DLog and DLEQ proof systems (Section 2.2) that are used for proving the exponentiations. The constructions themselves bear similarities to practical verifiable pseudorandom function protocols defined in prime-order groups (such as 2HashDH [11], [17], [22]).

Correctness. The correctness requirement for a VE (VEP) protocol is given in definition 3.1.

Definition 3.1 (VE/VEP Correctness)**.** *Let Γ be a protocol consisting of algorithms $(\mathsf{Gen}, \mathsf{Eval})$, let $\ell = \mathsf{poly}(\lambda)$, and let $\sigma \in \mathsf{Perm}[\ell]$. We say that Γ is a correct* VEP *protocol if the following inequality is satisfied:*

$$\Pr\left[A_{\sigma^{-1}(i)} \neq a_i^z \;\middle|\; \begin{array}{l}(z,Z,\pi_z)\leftarrow\Gamma.\mathsf{Gen}()\\ 1\leftarrow\mathsf{DLog.Verify}(Z,\pi_z)\\ ((A_{\sigma(i)})_i,\pi)\leftarrow\Gamma.\mathsf{Eval}((z,Z),(a_i)_i,\sigma)\\ \mathsf{DLEQ.Verify}((z,(a_i)_i),(Z,(A_{\sigma^{-1}(i)})_i),\pi)\end{array}\right] < \mathsf{negl}(\lambda).$$

Note that, when $\sigma = \mathsf{Id}$, then Γ is a correct VE *protocol.*

Security model. We now give the formal definition of security for a VE (VEP) protocol, which essentially amounts to showing that the protocol can be simulated in computational zero-knowledge, with respect to the evaluation server's secret data. Intuitively, this gives us the guarantee that the protocol computationally hides the server secret exponent used to evaluate the function. This security property is formalised in $\mathsf{Exp}_\Gamma^{\mathsf{x}}(\mathcal{A}, \ell, \lambda)$ for $\mathsf{x} \in \{\mathsf{VE}, \mathsf{VEP}\}$ in fig. 7, and Definition 3.2 captures the associated security condition.

Definition 3.2 (VE/VEP Simulation Security)**.** *Let Γ be a protocol consisting of algorithms $(\mathsf{Gen}, \mathsf{Eval})$. We say that Γ is a secure $\mathsf{x} \in \{\mathsf{VE}, \mathsf{VEP}\}$ protocol if, for all PPT algorithms $\mathcal{A}$, then $\mathsf{Adv}_{\Gamma,\mathcal{A}}^{\mathsf{x}}(\lambda, \ell) < \mathsf{negl}(\lambda)$ is satisfied, for all $\ell \in \mathsf{poly}(\lambda)$.*

Protocol guarantees. We now justify the correctness and security of the construction given in fig. 7.

Lemma 3.1 (Correctness of Γ)**.** *The protocol Γ defined in fig. 7 is correct, according to Definition 3.1.*

Proof. It's clear that $A_{\sigma^{-1}(i)} = a_i^z$ by the definition of $((A_i)_{i\in[\ell]}, \pi) \leftarrow \Gamma.\mathsf{Eval}(((z, Z), (a_i)_{i\in[\ell]}))$. Therefore, by the completeness of DLEQ, we know that π will verify correctly with probability 1. Furthermore, by the perfect completeness of DLog, we know that $1 \leftarrow \mathsf{DLog.Verify}(Z, \pi_z)$ with probability 1. As a result, the correctness of Γ follows. Clearly, this also holds in the case of VE, where $\sigma = \mathsf{Id}$. □

Lemma 3.2 (Security of Γ)**.** *The protocol Γ defined in fig. 7 is a secure* VE (VEP) *protocol, according to Definition 3.2, for $\ell \in O(\mathsf{poly}(\lambda))$.*

Proof. The proof of Lemma 3.2 follows in Appendix A. □

3.1. Handling Malicious Inputs

Programmable evaluation. During our eventual oblivious tally construction, we make use of a programmed evaluation mechanism for a VE/VEP protocol Γ. In essence, we require this to account for adversaries who have the ability to check that certain malicious entries are included in the tally result. In such cases, a standard simulation that randomises all outputs would not preserve the intersection result. We handle this functionality in the Γ.PEval function.

The Γ.PEval function is used internally during SEval, but in this case all inputs are sampled at randomly. However, the PEval provides the capability to provide a specific set of inputs $I = \{(i_j, A^*_j)\}_{i_j}$, where Γ is programmed to output A_i on input a_i, for any index $(i, A_i) \in I$. Note that, when we program m out of ℓ inputs, we essentially find ourselves in $\mathcal{H}_{3,m}$ of the proof of Claim A.0.3 during the proof of Lemma 3.2. As a result, as long as the programmed inputs A^*_j are indistinguishable from random group elements, it is clear that programmed evaluation is indistinguishable from fully-simulated evaluation.

Non-random inputs. The protocol shown in fig. 7 deliberately does not consider *"client-side"* guarantees, since we assume that the inputs $(a_i)_{i\in[\ell]}$ to the protocol are always sampled *i.i.d.* uniformly randomly in $\mathbb{G}$. In real protocols, clearly the client could cheat by providing non-random values to the protocol, and then checking whether the relationships are preserved. In the simulated case, because the secret exponent is not used such relationships are highly likely to be disturbed.

In our eventual protocol in Section 6, we target malicious security, which means that VE clients can deviate from the desired functionality in this way. However, we get around this by adding protocol-level checks and proofs of computation, that allow us to guarantee the inputs to the VE (VEP) protocols are always eventually sampled randomly. In the case where the client may attempt to deviate from the exchange, we simply instruct the protocol-level proof simulation to abort, which mirrors the checks that honest observers of the protocol can perform in the real-world. See Section 5.1 for further discussion on how we handle this in the protocol layer.

4. System Model

Assumed architecture. As discussed previously, our scenario and system model is motivated by the Signal protocol and the associated Signal E2EE messaging application [10]. As such we build reputation systems only assuming the existence of participants that are currently supported by Signal. Within this framework, we then would like to develop a minimal protocol (based on believable assumptions, practical cryptographic primitives and constructions), that enable our reputation framework to augment the existing application.

In the Signal protocol, there is a universe of users ($\mathcal{U}$), and a *single* application server (S). While this server could obviously be distributed across many geographically disparate nodes, the use of *"single"* here simply denotes that we do not make any non-collusion assumptions about the entity *controlling* the server (in this case, Signal). In this universe, users may communicate with each other directly over E2EE channels (via the server), or they may form *groups* of communicating individuals. In the existing protocol, such groups are either private — in which each access to the group (via a hyperlink) is managed by a specific administrator who must personally approve each request — or public — where anybody with a link can access the group. Clearly, all such requests (public or private) are routed through the application server. While Signal does not explicitly know the membership of each group, there are no actual guarantees that such information is not leaked during the protocol execution (including individual user contact graphs).

Voting overlay. Our first modification to the existing system, is to build a voting overlay that allows assigning reputation to any given user in the system. Each user can vote on any other user in the system. For this purpose, we can think of a user as divided into both a *voter* and *votee* entity, each with their own voter/votee key material. Voters create *ballots* on users corresponding to a plaintext *vote*, using their voting key, and the target user's votee key pair. These ballots must maintain the confidentiality of the vote itself, and are sent to and held by the server. In our main voting system in Section 6, we allow each U_i to vote on U_j *once*. Once a vote is cast, it remains permanently. In Section 9, we discuss an approach for allowing multiple votes to be cast by U_i on U_j, where only the most recent vote is counted. Our main voting system also assumes that the user themselves will store ballots cast on them by other users.

Semi-open groups. Our second modification is to introduce the concept of a *semi-open messaging group*. Such groups are useful in contexts where there is a desire to ensure that group membership is monitored beyond simply the sending of a link, but where individual manual vetting of all join requests is considered infeasible. The desire for groups appears to be very common, especially in activism, resistance, and organising contexts. Our solution involves using the previous voting overlay to build a reputation system by which to monitor entries into such groups. The following paragraph summarises the functionality of said groups.

The group (G) is made up of a collection of initial users ($\mathcal{V}|_{\mathsf{G}}$). These initial members are assumed to be hand-picked, and manually admitted into the group. Subsequent join requests made by users U_{i^*} are approved based on the reputation of U_{i^*} amongst the users $\mathsf{U}_j \in \mathcal{V}|_{\mathsf{G}}$ (i.e. derived from the ballot cast by each U_j on U_{i^*}, if one exists). The reputation score for an external user U_{i^*} attempting to join G is calculated by a single user who is deemed to be the group administrator (we abuse notation, and refer to this user as G). We assume that there is a function (Tally) that, given plaintext votes, calculates a value $k \in \mathbb{Z}$ that corresponds to a user's reputation. As such, we also assume

that there is a protocol Π that, given the ballots cast on U_{i^*}, allows G to calculate k. A *positive* reputation score (e.g. $k > \tau$, for some threshold τ) can be admitted to G automatically, or be subject to subsequent steps. A *negative* reputation score (e.g. $k \leq \tau$) may be rejected automatically, or optionally manually handled subsequently. Actions of G are monitored by each $\mathsf{U}_j \in \mathcal{V}|_{\mathsf{G}}$. Communication between entities is performed online via the application server S, but it is assumed that non-admin members in $\mathcal{V}|_{\mathsf{G}}$ may be *offline*. Therefore, their assistance during the protocol is not assumed. The only online parties are therefore G, U_{i^*}, and S.

In the following, we simplify this model slightly, and assume that a U_{i^*} with high enough reputation is automatically admitted into the group if the reputation score is high enough. In this case, $\mathsf{U}_{i^*} \rightarrow \mathcal{V}|_{\mathsf{G}}$, and learns all group parameters that existing group members have access to.[4]

Reputation score calculation. We leave the construction of an optimal choice for Tally for calculating the reputation scores as an open question, as well as the explicit strategy for choosing the threshold τ. In particular, our protocol can handle a Tally function that calculates sums of votes $x \in D$, where $|D| = \mathsf{poly}(\lambda)$. Alternative solutions may make more sense, depending on the context and application, and further research on such functions (and how they could be embedded in such protocols) would be highly valuable.

5. Security Model

We now discuss the overall security model for protocols automating admissions into semi-open groups. In fact, our approach models security in any reputation system/e-voting scenario, given the system design assumed in Section 4.

Ideal functionality. As discussed in Section 2.4, we prove security in the standalone MPC real-/ideal-world paradigm, for ensuring security against any corrupted (malicious) entity in the protocol. We define the ideal functionality available to the simulation in the security proof in fig. 8. The strong-variant of the ideal functionality $\mathcal{F}$ computes the ideal tally function (Tally) over the provided honest input votes from all users $\mathsf{V}_j \in \mathcal{V}|_{\mathsf{G}}$, on the target user U_{i^*}. In the weak variant of $\mathcal{F}$, the set of all collected votes is released to the simulator directly, who can then calculate Tally themselves. During the simulated protocol, the simulator will attempt to extract a valid vote from any malicious voter, and then run $\mathcal{F}$ including this vote, or $\perp$ if no vote is provided (or if all voters are honest).

We provide the strong variant for context and future work, while we use the weak variant when writing the security proof. Intuitively, the reason why the strong variant cannot be satisfied in the eventual protocol (Section 6) is due to the fact that the real protocol intentionally unmasks each individual element in the intersection. A stronger protocol would perform the intersection blindly and the computation of Tally blindly, before revealing the result b.

4. This is an important consideration for security: we need to ensure that group-specific parameters cannot be used by a malicious U_{i^*}, who gains access to the group, to open individual ballots that they hold.

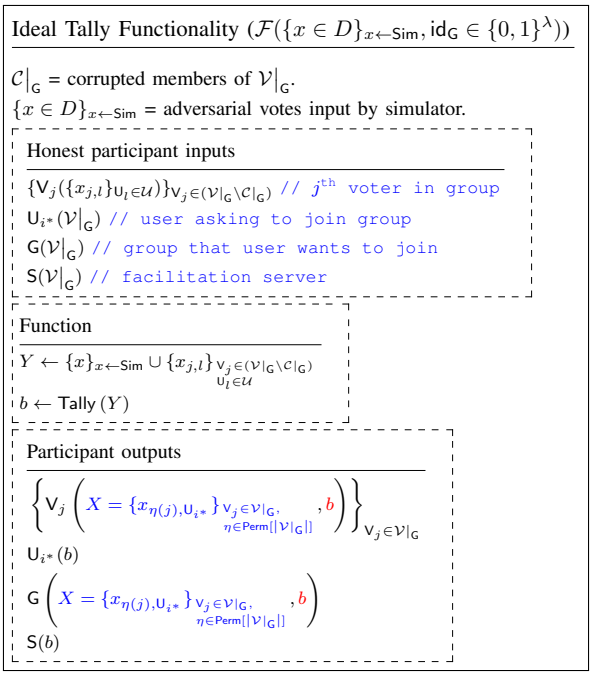

Figure 8. (Weak/Strong) Ideal Functionalities

Corruptions, simulations, aborts. To prove security, we must construct a series of corruption models, whereby $\mathcal{A}$ corrupts some subset of the entities in the protocol, and then show that there exists a PPT simulator for each such corruption model. In all cases, we assume corruptions by a malicious adversary, that are selectively defined apriori to the functioning of the protocol. We note that the types of corruptions we permit strengthen or weaken the security model. For example, proving security against an adversary that only corrupts one of $\{\mathsf{U}_{i^*}, \mathsf{G}, \mathsf{S}, \mathcal{C}|_{\mathsf{G}} \subset \mathcal{V}|_{\mathsf{G}}\}$ is weaker than one that corrupts combinations of these entities at once. In the base version of the protocol in Section 6, we prove security against an adversary that either corrupts the set $\{\mathsf{U}_{i^*}, \mathsf{G}, \mathcal{C}|_{\mathsf{G}} \subset \mathcal{V}|_{\mathsf{G}}\}$ or S alone. In the former, we essentially show that a malicious combination of the group admin, some subset of group members, and the target user cannot successfully subvert the protocol, beyond simply submitting enough votes to impact the intersection. In other words, we show that the protocol remains demonstrably secure for honest group members who witness the protocol transcript. Note that proving security in this case transitively demonstrates security for an adversary that corrupts only subsets of these individuals (e.g. in the case where U_{i^*} is honest). In the latter case, we demonstrate security against a malicious server alone. Note that in Section 8.1, we demonstrate a protocol adaptation that allows us to prove security against an adversary that corrupts both S, and one other protocol entity.

To guarantee security, we then prove that the real protocol is indistinguishable from an ideal-world simulation,

for each corruption model that is considered. Note that the protocol at any point can be aborted arbitrarily by the adversary, who sends a message to the simulator, and thus we in fact prove malicious security *in the presence of aborts*.

Random oracles. Throughout the protocol, we assume that all adversarial random oracle calls (via the function $\mathsf{RO} : \{0,1\}^{\ell_{\text{in}}} \mapsto \{0,1\}^{\ell_{\text{out}}}$) are simulated internally by the simulator. For new queries q seen by the simulator, they sample a random response as $\mathsf{r} \leftarrow\$ \{0,1\}^{\ell_{\text{out}}}$, where ℓ_{out} is the bit-length of the output, and then add (q, r) to a table, and return r to the adversary. For already observed queries, the simulator recovers the pair (q, r), and returns r to the adversary. We denote such interactions by $\mathsf{r} \leftarrow \mathsf{RO}(\mathsf{q})$.

5.1. Assumptions

With the system and security models covered, we now clarify a number of assumptions that we make for simplifying the security argument of the protocol. We believe all such assumptions to be realistic, given real-world implementations of secure messaging systems.

Internal group transcript. During the protocol defined in fig. 9, it's clear that parties send and receive messages from each other. In the case of the group admin G, messages received from other (honest) parties are assumed to be directly written to the internal transcript of the group. This is important, since the honest voters in the group use the internal group transcript to decide whether the protocol is executing as expected. This assumption therefore excludes malicious groups that can arbitrarily modify this transcript. We argue that this assumption is believable in settings where honest external parties sign their network messages with long-term public keys. Such a system would require an internal PKI, and so we avoid describing such a network to maintain the simplicity of the overall protocol design.

Retrospective aborts. During the protocol execution, we must allow for users inside the group to be offline. This matches the real-world scenario, where a user (U_{i^*}) may request to join, be reviewed, and finally be added to the group, all while a voter within the group remains offline. Any vote that $\mathsf{U}_j \in \mathcal{V}|_{\mathsf{G}}$ produced on U_{i^*} would still be counted. However, in the case of an adversarial G, there are various steps in the protocol that require honest users in the group to ratify the actions of G. If G deviates from the protocol specification, then it is assumed that such users will call `abort` and the protocol will be aborted. During the security proof, we assume that such voters are online at the moment of aborting. However, in the real-world implementation, users that return online must replay executed protocol steps and check that everything matches their view of the group. If they find that G acted incorrectly, then said user must trigger such aborts *retrospectively*, and effectively act as a whistleblower on G.

Formally speaking, we maintain *eventually malicious* security. In essence, the protocol may be abused by a malicious party, and may continue for some amount of time afterwards. However, once an honest party that verifies the malicious behaviour comes online, the protocol instance will be abandoned (either during, or afterwards). We note that in the time between malicious behaviour occurring and an abort being triggered, no guarantees are given about the real and simulated worlds being indistinguishable.

Ballot ordering. The protocol in Section 6 assumes that ballots cast on U_{i^*} are collected and held by both U_{i^*} and S. It is assumed that both parties hold ballots in some random, independent ordering. While in the real-world, such orderings are likely to be dictated by the order that votes are received, this is merely a simplifying assumption to make the simulation more streamlined.

Polynomial-sized vote domain. Our proposed scheme relies on encryption "in the exponent" technique, where a vote $x_{i,j}$ from voter V_i to user U_j is encrypted as $c = g^{v_i u_j} \cdot g^{x_{i,j}}$. To decrypt, one needs to compute the discrete logarithm $\log_g(c \cdot g^{-v_i u_j}) = \log_g(g^{x_{i,j}})$. This is possible because we constrain the space of valid votes to a small set D of polynomial size, so that the logarithm can be efficiently computed by exhaustive search in $\{g^x \mid x \in D\}$. More advanced "blind intersection" techniques could potentially allow circumventing this limitation, while achieving the stronger ideal functionality in Figure 8. A possible idea would be that of identifying an efficient encoding $\varepsilon_{\mathbb{G}}(\cdot)$ for fully-homomorphic ciphertexts into a large group $\mathbb{G}$, so that $g^{x_{i,j}}$ is replaced by $\varepsilon_{\mathbb{G}}(\mathsf{FHE.Enc}(x_{i,j}))$. This however would likely require very large groups and key material, and a slow intersection algorithm. For this reason, we leave blind intersection techniques to potential future work (Section 9).

6. Protocol Design

We recall crucial notation and conventions. Algorithms and entity identifiers are written in camel-case sans-serif font. Protocol entities are: S (Server), G (Group admin), U_i (i^{th} user in system), U_{i^*} (user attempting to join group). Sets of entities are written in calligraphic font, such as: $\mathcal{U}$ (all users in ecosystem), $\mathcal{V}|_{\mathsf{G}}$ (all voting users belonging to the group), $\mathcal{V}|_i$ (all users voting on U_i). For arrays of group elements, we use capital-case (e.g. Z in $\mathsf{G.AddUser}$ is the set of tags submitted to G). We write Votes_{i^*} to denote the set $\left\{g^{u_{i^*} v_j + x_{j,i^*}} \middle| \mathsf{U}_j \in \mathcal{V}|_{i^*}\right\}$ of anonymised votes received by U_{i^*}. This is held both by U_{i^*} and S. Additionally, we assume that all NIZK proofs are assumed to be verified by parties who receive them. For a set of group elements, $Y \in \mathbb{G}^\ell$, we write Y^s (for $s \in \mathbb{F}(\mathbb{G})$) to denote the set $\{y^s\}_{y \in Y}$.

Protocol specification. In fig. 9, we give the formal description of each of the algorithms used in the protocol that was detailed across Figures 1 to 5. In particular, the steps of the protocol are identical to the explanation in Section 1.1, except for the added zero-knowledge proofs that enable us to achieve security against malicious adversaries by forcing parties to prove they performed their role correctly. We add annotations to indicate explicitly where an algorithm may impact the view of another entity in the

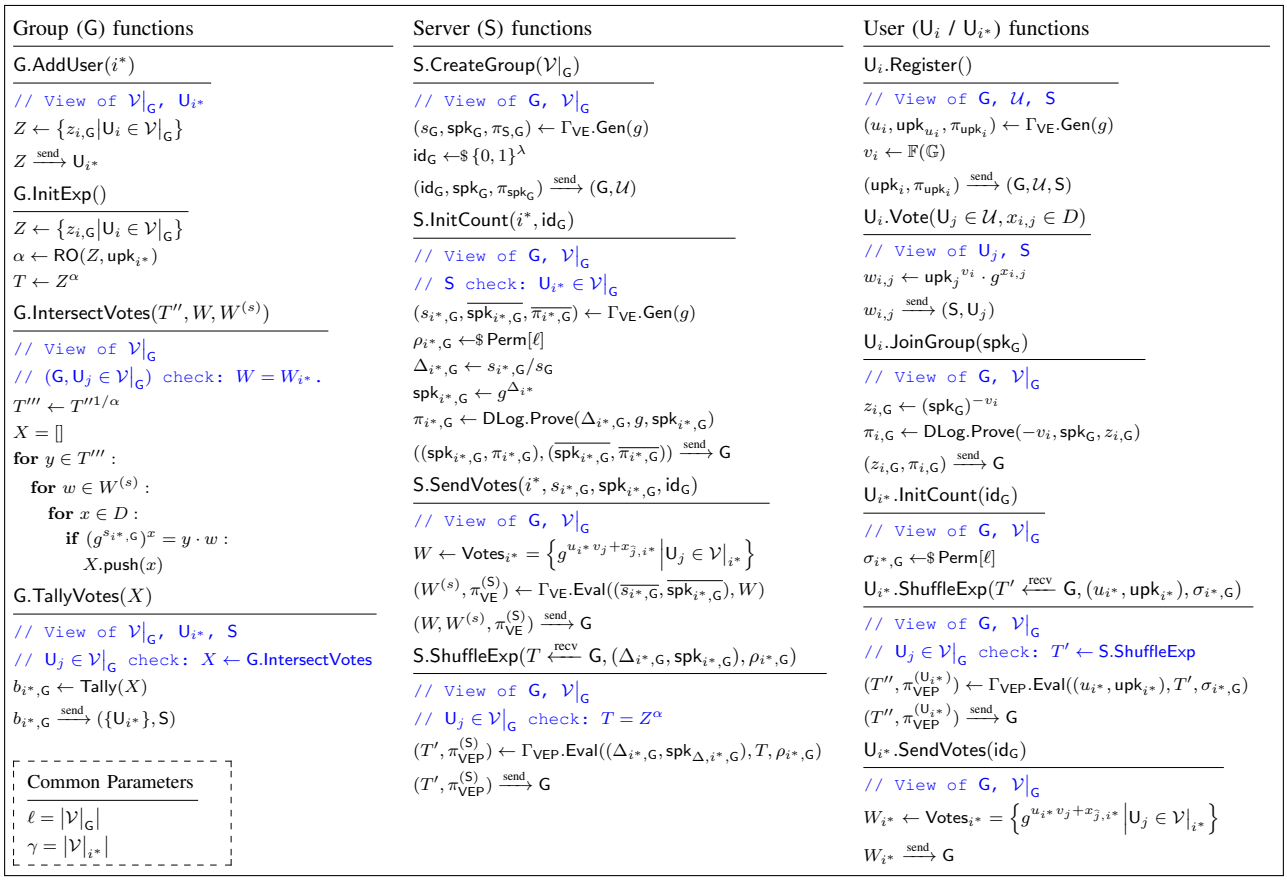

Figure 9. Full list of functions used in base protocol (Section 1.1).

protocol, which is pertinent for the simulation in the security proof.

6.1. Security Guarantees

We consider a target group G, where U_{i^*} is attempting to join. As mentioned in Section 5, we provide security against malicious adversaries corrupting various subsets of entities. In particular, we show that this protocol maintains security against $\mathcal{A} = (\mathcal{A}_{\mathsf{G}}, \mathcal{A}_{\mathsf{U}_{\hat{j}}}, \mathcal{A}_{\mathsf{U}_{i^*}})$ — i.e. an adversary that corrupts group admin, a subset of internal group members, and the target user — and $\mathcal{A} = \mathcal{A}_{\mathsf{S}}$ — a corrupted server. The main security guarantee and required cryptographic assumptions are elucidated in Theorem 6.1 below.

Theorem 6.1. *The base protocol described in Figure 9 is secure against a malicious adversary that corrupts* either *any subset of entities associated with a given group* G *(Lemma B.7),* or *the facilitation server (Lemma B.8). This follows under the existence of a secure* VEP *protocol (Definition 3.2), the hardness of DDH, and a NIZK proof of knowledge system for discrete log relations.*

Proof. For the full formal security proof of each corruption case, we refer the reader to Appendix B. □

Remark 6.1. *For handling stronger corruption models (including server collusion), Section 8.1 describes an adapted protocol sharing group admin responsibilities amongst $n > 1$ non-colluding group admins. This allows for the handling of corruptions of the form $\mathcal{A} = (\mathcal{A}_{\mathsf{S}}, \mathcal{A}_{\mathsf{X}})$, for $\mathsf{X} \in \{\mathsf{G}, \mathsf{U}_{\hat{j}}, \mathsf{U}_{i^*}\}$, where a malicious server colludes with either the external user, or a group admin or member. The security argument follows naturally after splitting the responsibilities of the group admin amongst non-colluding members, as we sketch in Section 8.1.*

Intuitive guarantees. Regarding the corruption cases in Theorem 6.1, the first shows that the base protocol is robust to a malicious subset of group members $(\mathsf{G} \cup (\mathcal{C}|_{\mathsf{G}} \subset \mathcal{V}|_{\mathsf{G}}))$ and the target user (U_{i^*}) attempting to subvert the reputation calculation, to either force unwanted group entries, or deanonymise votes. Note that this immediately implies security against an adversary that corrupts only a subset of these entities as well. In the case of the server, corruptions are harder to handle, and thus we assume no further collusion with other entities in the group (or who may join the group). We discuss the reasons for this in the following, where we highlight a series of *intuitive* security properties that are maintained by virtue of our proof technique.

Vote confidentiality. Our protocol ensures vote confidentiality for all votes. That is, for any given ballot in

the system on U_{i^*}, an adversary cannot discern the value of said vote apart from based on the information that is leaked by $\mathcal{F}$ (as in all MPC protocols). Intuitively, this follows as a consequence of the fact that for any user U_j voting on U_{i^*}, the ballot takes the form: $g^{u_i v_j + x_{j,i}}$. Meanwhile, in the group, the voter tag takes the form $z_j = g^{-s_\mathsf{G} v_j}$. For any $\mathcal{A}$ that does not involve S, then we have an effective DDH relation of the form $(g, g^{1/s_\mathsf{G}}, z_j, g^{v_j})$ (in fact, g^{1/s_G} is never actually known), and thus this ensures that g^{v_j} is essentially indistinguishable from uniform, which masks the vote $x_{j,i}$. Note that this does not take into account votes that are eventually learned in the intersection of voting users in G on U_{i^*}, but the combination of VEP protocols means that the exponentiated tags in T''' are sufficiently shuffled to prevent leaking the identity-vote correlation. In the setting where S is corrupted alongside some entity in (or that eventually joins G), this problem becomes easy to solve, as $\mathcal{A}$ is able to send z_j to the server, who can then remove s_G from the voter tag to learn g^{v_j}, and thus $x_{j,i}$.

Ballot unlinkability. For the reasons described above, it is worth noting that ballots themselves are actually unlinkable from the identities of the users casting them. In particular, this is since each ballot is distributed uniformly in $\mathbb{G}$, even given the voter tags held by the group. Therefore, assuming there is an anonymous channel for submitting such ballots, the identities of the users themselves are also hidden (again, accepting the leakage of $\mathcal{F}$ with respect to the group G).

Tally integrity. We maintain the integrity of the tally computed by G by allowing access to all internal group parameters to entities in $\mathcal{V}|_\mathsf{G}$. Since we assume that all communications between group admin, external user and the server are written to the internal group transcript, internal users can then follow the protocol, verify the various zero-knowledge proofs, and recompute the output of $\mathsf{Tally}(X)$ themselves (where X is the set of votes learned in the inetrsection function). Transitively, by maintaining the integrity of the tally, we also maintain the guarantee that any G that admits/does not admit a user incorrectly will be caught by honest group members.

7. Implementation and Benchmarking

In order to verify the practical viability of our protocol, we implemented the DLog and DLEQ proof systems in Section 2.3, the verifiable exponentiation protocols in Section 3, and the base protocol from Section 6 (including all algorithms from fig. 9). To instantiate the protocol, we use the Ristretto255 prime-order group [13]. Random oracles within proof systems are implemented using SHA2-256, with the exception of the shuffled DLEQ proof system that uses SHAKE128.[5] We separate the oracles across proof creations by pre-pending a 256-bit random prefix to the input, independently sampled during each proof.

Our implementation is written in C++, and is available at https://github.com/luizabrs/semi-open-messaging-groups.

5. This is for simplicity, as a challenge longer than 256 bits is used.

Parameters	Phase	Runtime (s) mean	st. dev.	Bandwidth (KiB)
$n = 50$ $t = 40$ $\lvert D\rvert = 10$	total	3.3	0.2	1312.2
	VE.Eval & check	< 0.1	< 0.1	2.6
	VEP.Eval & check (U)	1.2	0.1	653.2
	VEP.Eval & check (S)	1.2	0.1	653.2
	ballot intersection	0.9	0.1	1.2
$n = 100$ $t = 40$ $\lvert D\rvert = 10$	total	6.4	0.4	2620.1
	VE.Eval & check	< 0.1	< 0.1	2.6
	VEP.Eval & check (U)	2.2	< 0.1	1306.3
	VEP.Eval & check (S)	2.3	0.4	1306.3
	ballot intersection	1.9	0.0	1.2
$n = 200$ $t = 40$ $\lvert D\rvert = 10$	total	12.7	0.2	5235.7
	VE.Eval & check	< 0.1	< 0.1	2.6
	VEP.Eval & check (U)	4.5	0.1	2612.5
	VEP.Eval & check (S)	4.5	0.2	2612.5
	ballot intersection	3.7	0.0	1.2
$n = 200$ $t = 80$ $\lvert D\rvert = 10$	total	16.3	0.2	5239.4
	VE.Eval & check	< 0.1	< 0.1	5.1
	VEP.Eval & check (U)	4.5	0.1	2612.5
	VEP.Eval & check (S)	4.4	0.1	2612.5
	ballot intersection	7.4	0.1	2.5

TABLE 1. COSTS OF VARIOUS PHASES OF THE BASE PROTOCOL BETWEEN A GROUP OF n USERS AND AN EXTERNAL USER THAT RECEIVED t VOTES, AVERAGED OVER 10 RUNS OF THE PROTOCOL.

We use Ristretto and SHA2 implementations from `libsodium` [14] (`v1.0.20`), and the SHAKE implementation from the "compact" FIPS202 code in the XKCP library [5]. Our code models a scenario where $n + t/2 + 1$ users U_i and one group G are registered with the server, and users $\mathsf{U}_1, \ldots, \mathsf{U}_n$ are manually added to G. Then, t users, half inside the group and half outside of it, send votes from a domain D on $\mathsf{U}_{n+t/2+1}$. Finally, a request from $\mathsf{U}_{n+t/2+1}$ to join G is simulated, where the t ballots are intersected with the n tokens held by G. We recall that performance is independent of the size of the entire universe of users, so our experiments focus only on varying group sizes.

Within the protocol, most of the time is spent either during VEP evaluation and verification or during ballot intersection. While both VE and VEP run in time $O(n)$, VEP requires many more group operations due to the generic compiler [20] used to prove a correct shuffle. Ballot intersection runs in time $O(n \cdot t \cdot |D|)$. These operations are amenable to trivial parallelisation speedups. In Table 1, we report clock times on evaluations of the protocol and its components, for various group sizes n and numbers of ballots t, and ephemeral communication costs.[6] All measurements are performed on a Macbook Air M3 CPU on a single core. Overall, we see that a naïve implementation achieves acceptable runtimes for relatively large values of n and t, and shuffled DLEQ soundness error 2^{-128}.

6. We don't count costs for transferring long-term public keys that are generated only once, or single ballots cast on a user before the join request.

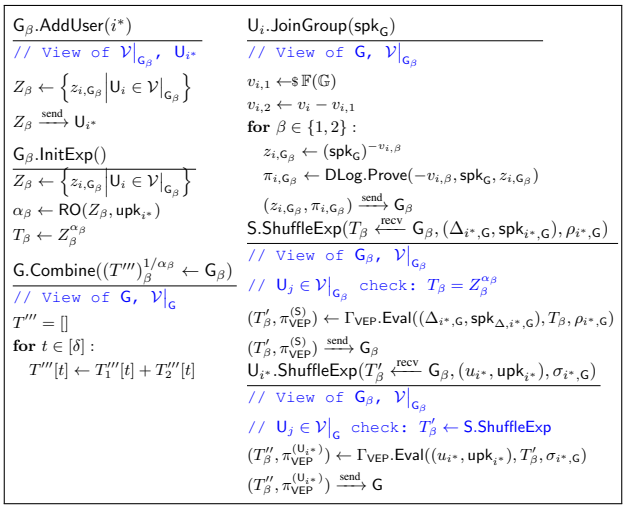

Figure 10. Modified algorithms for adapted (collusion-resistant) protocol.

8. Protocol Adaptations

We now present two adaptations of the base protocol, that handle limitations regarding a colluding adversary that corrupts the server and one other entity (Section 8.1), as well as removing the restriction that any user can only vote once on another user (Section 8.2).

8.1. Handling a Colluding Server

When the adversary corrupts the server in the main protocol (Section 6), we cannot tolerate corruption of any other entity associated with the group. Otherwise, this leads to the possibility of trivially deanonymising votes made on the corrupted user. This is a fundamental problem with the protocol's system model: Voter tags are generated without knowledge of U_{i^*}, ballots without knowledge of G, and the only always-known shared object is S's public key. Therefore a colluding server is hugely damaging.

To rectify this issue, we consider an adapted protocol that can handle corruptions that involve S and some entity $\mathsf{X} \in \{\mathsf{G}, \mathsf{U}_{i^*}, \mathcal{C}|_{\mathsf{G}} \subset \mathcal{V}|_{\mathsf{G}}\}$. The adapted protocol considers two group admins $(\mathsf{G}_1, \mathsf{G}_2)$. In this case, we assume that *at least* one of the admins is honest.[7] Furthermore, each group member secret-shares their voter tag (in the exponent) amongst both admins, and finally each member is assigned to the view of only one single admin (i.e. $\mathsf{U}_{\hat{j}}$ belongs to one of $\mathcal{V}|_{\mathsf{G}_\beta}$ for $\beta \in 1, 2$). This means that each member only learns "half" of the tag for each other group member.

Let G refer to the group as a whole, while G_β refers to a specific admin. We give the modified protocol algorithms in Figure 10. The main differences to highlight are that both group admins essentially execute the same protocol as before, but interacting with S and U_{i^*} separately (using $\alpha_\beta \leftarrow \mathsf{RO}(Z_\beta, \mathsf{upk}_{i^*})$ as the blind for their share of the voter tags). Finally, in the G.Combine algorithm, both group admins share the sets of exponentiated shuffled tags with each other (noting that both sets are shuffled identically). Finally, the rest of the intersection protocol advances in the same manner as before.

Sketch security argument. Without loss of generality we focus on the case where some elements in G_1 are corrupted. Specifically, consider an adversary $\mathcal{A}$ that corrupts S, and one of G_1, $\mathcal{C}|_{\mathsf{G}_1} \subset \mathcal{V}|_{\mathsf{G}_1}$, or U_{i^*}. Security will essentially follow because no entity associated with G_1 (either the admin themselves, the voters assigned to $\mathcal{V}|_{\mathsf{G}_1}$, or U_{i^*} if they are admitted) will ever see a full voter tag for any honest group member. Therefore, the secret voting exponents used by such members will only be exposed in the votes they cast, and in the shuffled exponentiated lists of tags after Combine. By definition, since we assume that $\mathcal{A} = (\mathcal{A}_\mathsf{S}, \mathcal{A}_\mathsf{X})$, then there will always be at least one party honest in at least one ShuffleExp exchange (in particular, either G or U_{i^*}). This primarily allows simulation of the ShuffleExp exchange for the adversary, without exposing the voting exponent. After, it's simple to show that the votes alone are distributed uniformly, and thus hide the exponent. Lastly, we note that while $\mathcal{A}_{\mathsf{G}_1}$ can choose to send an incorrect T'''_1 to G_2, our assumption that at least one entity in both $\mathcal{V}|_{\mathsf{G}_\beta}$ is honest prevents this from leading to incorrect results in the protocol.

8.2. Updating votes

As noted previously, our protocol only tolerates a voter casting a single vote on any given user. This is due to the fact that, for votes $x^{(1)}_{j,i}$ and $x^{(2)}_{j,i}$, then $g^{u_i v_j + x^{(1)}_{j,i}} / g^{u_i v_j + x^{(2)}_{j,i}} = g^{x^{(1)}_{j,i} - x^{(2)}_{j,i}}$, i.e. the difference of the plaintext votes. This restriction may not be a problem for many applications, since a voter that wants to update votes can simply create new voter parameters (i.e. secret exponents by rerunning Register), and then vote again. However, if we want to be able to update individual votes without re-registration, we must consider changes to the protocol. In order to handle multiple votes, we believe an alternative protocol where votes are constructed as 2-tuples of the form $(g^{u_i v_j}, g^{H(v_j, l) + x^{(l)}_{j,i}})$ (where H is modelled as a random oracle in $\mathbb{F}(\mathbb{G})$) gives us the required security guarantee. Notice here that the counter $l \in \mathbb{N}$ is used to domain separate the hashes of each vote, where l is incremented for each vote update. In practice, l must be bounded by some $B \in \mathbb{N}$ so that attempted intersection of the votes can be performed by the group. In principle, this scheme actually requires the voters to upload the tags $Z_j = \{g^{s_\mathsf{G} H(v_j, l)}\}_{l \in [B]}$ to each group that they joins. By modifying VEP to operate over and shuffle sets of tags, we can still perform the intersection, and we also can ensure that only the most recent vote is counted.

In terms of a brief security analysis, the first element of the vote $(g^{u_i v_j})$ guarantees that each ballot (and its updates) are linkable, as in the linkable ring signature approach of AnonRep [33]. Secondly, ratios of the second element still hide the plaintext vote. However, one thing to notes is that each of the entire sets Z_j must be exponentiated by S and U_{i^*} during the VEP interactions. We do not require that

7. We allow n admins with one honest, but consider two for simplicity.

these sets maintain their order, and so after these interactions it is assumed that all of the tags are shuffled together. In any case, since the original protocol permits vote updates via reregistration, we stop short of offering a full analysis of this variant, which we leave to future work.

9. Limitations and Future Work

We focus on some open problems associated with our work, that could be the focus of valuable future work. Note that addressing the limitations described in Section 1.2 would appear to violate the practicality constraints that we enforce in the system model (Section 4).

Currently, we lack an obvious path to building a post-quantum (PQ) version of the protocol. The main hurdle would appear to be finding practical solutions to the problems of building VEP protocols, and the development of linkable ring signature-like primitives for casting votes. While recent works on OPRFs match the format of the 2HashDH exponentiation interface that we use [22], none of them appear remotely close to practical deployment [2], [3], [6]. Any innovations in this space would design a clearer path to a PQ alternative of our protocol. A second limitation is that we only satisfy security with respect to the weak formulation of the ideal functionality in fig. 8, since all (shuffled) votes are revealed in the clear. A protocol that reveals only the tally result would be a welcome improvement, as noted in [18] where it is highlighted that only revealing thresholds on reputations is an obvious privacy enhancement. Such a protocol would likely require usage of heavier primitives (such as FHE) or more interaction, and so research into practical solutions to this problem would be highly welcome. Finally, further investigation into more complex tallies and how they interact with the overall system would be useful. This would potentially allow building generic reputation systems that permit more nuanced operations and calculations.

Acknowledgements

The authors would like to thank Kevin Gallagher for useful discussions on the topic, and the organizers and attendees at CAW (co-located with IACR Eurocrypt 2024), where a preliminary version of this work was presented. Some of the work was produced while Davidson and Virdia were affiliated with NOVA LINCS, Universidade NOVA de Lisboa. The research of Davidson was also supported by Fundação para a Ciência e a Tecnologia (FCT) through the LASIGE Research Unit (UIDB/00408/2020 and UIDP/00408/2020), and by NOVA.ID.FCT through the CertiCoLab project funded by the EEA Grants Bilateral Fund (FBR_OC2_103-CertiCoLab). The research of Virdia was also supported by UKRI grant EP/Y02432X/1.

References

[1] M. R. Albrecht, J. Blasco, R. B. Jensen, and L. Mareková. Mesh messaging in large-scale protests: Breaking Bridgefy. In *CT-RSA 2021*, volume 12704 of *LNCS*. Springer, Cham, May 2021.

[2] M. R. Albrecht, A. Davidson, A. Deo, and D. Gardham. Crypto dark matter on the torus - oblivious PRFs from shallow PRFs and TFHE. In *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 447–476. Springer, Cham, May 2024.

[3] M. R. Albrecht, A. Davidson, A. Deo, and N. P. Smart. Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. In *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 261–289. Springer, Cham, May 2021.

[4] C. Bell and S. Eskandarian. Anonymous complaint aggregation for secure messaging. *Proc. Priv. Enhancing Technol.*, 2024(3), 2024.

[5] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. Compact FIPS202. https://tinyurl.com/compact-fips.

[6] D. Boneh, D. Kogan, and K. Woo. Oblivious pseudorandom functions from isogenies. In *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 520–550. Springer, Cham, Dec. 2020.

[7] D. Chaum. Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA. In *EUROCRYPT'88*, volume 330 of *LNCS*. Springer, Berlin, Heidelberg, May 1988.

[8] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO'92*, volume 740 of *LNCS*. Springer, Berlin, Heidelberg.

[9] S. S. M. Chow, C. Ma, and J. Weng. Zero-knowledge argument for simultaneous discrete logarithms. In *Computing and Combinatorics, 16th Annual International Conference, COCOON 2010. Proceedings*, volume 6196 of *LNCS*. Springer, 2010.

[10] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the Signal messaging protocol. *Journal of Cryptology*, 33(4):1914–1983, Oct. 2020.

[11] A. Davidson, A. Faz-Hernandez, N. Sullivan, and C. A. Wood. Oblivious pseudorandom functions (oprfs) using prime-order groups. RFC 9497, RFC Editor, December 2023.

[12] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, July 2018.

[13] H. de Valence, J. Grigg, M. Hamburg, I. Lovecruft, G. Tankersley, and F. Valsorda. The ristretto255 and decaf448 groups. RFC 9496, RFC Editor, December 2023.

[14] F. Denis. libsodium. https://github.com/jedisct1/libsodium.

[15] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *USENIX Security 2004*, pages 303–320. USENIX Association, Aug. 2004.

[16] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, volume 263 of *LNCS*. Springer, Berlin, Heidelberg, Aug. 1987.

[17] S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Vcelak. Verifiable random functions (vrfs). RFC 9381, RFC Editor, August 2023.

[18] S. Gurtler and I. Goldberg. SoK: Privacy-preserving reputation systems. *PoPETs*, 2021(1):107–127, Jan. 2021.

[19] S. Gurtler and I. Goldberg. PRSONA: private reputation supporting ongoing network avatars. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society, WPES2022*. ACM, 2022.

[20] T. Haines and J. Müller. A novel proof of shuffle: Exponentially secure cut-and-choose. In *ACISP 21*, volume 13083 of *LNCS*, pages 293–308. Springer, Cham, Dec. 2021.

[21] R. Henry and I. Goldberg. Batch proofs of partial knowledge. In *ACNS 13International Conference on Applied Cryptography and Network Security*, volume 7954 of *LNCS*, pages 502–517. Springer, Berlin, Heidelberg, June 2013.

[22] S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253. Springer, Berlin, Heidelberg, Dec. 2014.

[23] F. Kerschbaum. A verifiable, centralized, coercion-free reputation system. In *Proceedings of the 2009 ACM Workshop on Privacy in the Electronic Society, WPES 2009*. ACM, 2009.

[24] F. Kerschbaum, J. Haller, Y. Karabulut, and P. Robinson. Pathtrust: A trust-based reputation service for virtual organization formation. In *Trust Management, 4th International Conference, iTrust 2006, Proceedings*, volume 3986 of *LNCS*. Springer, 2006.

[25] R. Khare and A. Rifkin. Weaving a web of trust. *World Wide Web J.*, 2(3):77–112, 1997.

[26] J. Liu and M. Manulis. pRate: Anonymous star rating with rating secrecy. In *ACNS 19International Conference on Applied Cryptography and Network Security*, volume 11464 of *LNCS*, pages 550–570. Springer, Cham, June 2019.

[27] J. K. Liu and D. S. Wong. Linkable ring signatures: Security models and new schemes. In O. Gervasi, M. L. Gavrilova, V. Kumar, A. Laganà, H. P. Lee, Y. Mun, D. Taniar, and C. J. K. Tan, editors, *Computational Science and Its Applications - ICCSA 2005, International Conference, Singapore, May 9-12, 2005, Proceedings, Part II*, volume 3481 of *LNCS*, pages 614–623. Springer, 2005.

[28] M. Marlinspike and T. Perrin. The X3DH key agreement protocol. *Open Whisper Systems*, 283(10), 2016.

[29] U. M. Maurer. Modelling a public-key infrastructure. In *ESORICS'96*, volume 1146 of *LNCS*. Springer, Berlin, Heidelberg, Sept. 1996.

[30] W. Müller, H. Plötz, J. Redlich, and T. Shiraki. Sybil proof anonymous reputation management. In A. Levi, P. Liu, and R. Molva, editors, *4th International ICST Conference on Security and Privacy in Communication Networks, SECURECOMM 2008, Istanbul, Turkey, September 22-25, 2008*, page 7. ACM, 2008.

[31] M. Thomson and C. A. Wood. Oblivious HTTP. RFC 9458, RFC Editor, January 2024.

[32] A. C.-C. Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*. IEEE Computer Society Press, Nov. 1982.

[33] E. Zhai, D. I. Wolinsky, R. Chen, E. Syta, C. Teng, and B. Ford. Anonrep: Towards tracking-resistant anonymous reputation. In *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016*. USENIX Association, 2016.

Appendix A. Proof of Lemma 3.2

To prove Lemma 3.2, we show that the case of $b = 0$ in the security model of fig. 7 can be transformed into something computationally indistinguishable from the case of $b = 1$, via a sequence of computationally indistinguishable hybrid steps. We detail the hybrid steps below, and then prove each of them individually in the sequence of claims that follow. Note that the proof applies in either the case of VE or VEP using the shuffled DLEQ proof construction mentioned in Section 2.3. This is because we eventually prove indistinguishability with a simulator that has no knowledge of the random permutation used in the VEP case. Therefore, without loss of generality, we prove the following hybrid transitions are indistinguishable with respect to VE.

Hybrid transitions

$\mathcal{H}_0$: This is the case standard security game (fig. 7).
$\mathcal{H}_1$: The case of $b = 0$ is modified to replace DLEQ.Prove with DLEQ.SProve.
$\mathcal{H}_2$: We replace DLog.Prove with DLog.SProve in Γ.Gen.
$\mathcal{H}_3$: We replace $A_i \leftarrow a_i^z$ in Γ.Eval with $A_i \leftarrow\!\$\, \mathbb{G}$.
$\mathcal{H}_4$: We replace $Z \leftarrow g^z$ with $Z \leftarrow\!\$\, \mathbb{G}$ in Γ.Gen.

Note that in $\mathcal{H}_4$, the case of $b = 0$ is exactly equivalent to the case of $b = 1$, and we are done. We now proceed to prove the indistinguishability of the transitions between $\mathcal{H}_0$ and $\mathcal{H}_4$, in Claims A.0.1 to A.0.4. We use $\mathsf{Adv}^{\mathrm{c}}_{\mathcal{D}}(\lambda)$ to denote the advantage of a PPT distinguishing algorithm $\mathcal{D}$ distinguishing between $\mathcal{H}_{\mathrm{c}-1}$ and $\mathcal{H}_{\mathrm{c}}$, for $c \in \{1, 2, 3, 4\}$.

Claim A.0.1. $\mathsf{Adv}^{1}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathrm{nizk}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda)$

Proof. Let $\mathcal{B}$ be a PPT adversary against the NIZK property of DLEQ. Assume that $\mathcal{B}$ simulates VE by instantiating DLEQ.Prove by sending all queries to the instances of the oracles $P_{b'}, R_{b'}$ that they have access to, for $b' \in \{0, 1\}$. In the case of $b' = 0$, this is exactly the same as in $\mathcal{H}_0$, because P_0 simply outputs the same as DLEQ.Prove. In the case of $b' = 1$, P_1 outputs DLEQ.SProve, which is the same as $\mathcal{H}_1$. Therefore, if $\mathcal{D}$ distinguishes between $\mathcal{H}_0$ and $\mathcal{H}_1$ with advantage ϵ, this can be turned into an immediate distinguisher for $\mathsf{Exp}^{\mathsf{nizk}}_{\mathsf{DLEQ}}(\mathcal{B})$. □

Claim A.0.2. $\mathsf{Adv}^{2}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathrm{nizk}}_{\mathsf{DLog},\mathcal{B}}(\lambda)$

Proof. Let $\mathcal{B}$ be a PPT adversary against the NIZK property of DLog. Assume that $\mathcal{B}$ simulates VE by instantiating the usage of DLog.Prove by sending all queries to the instances of the oracles $P_{b'}, R_{b'}$ that they have access to in the NIZK security game, for $b' \in \{0, 1\}$. Note that this leads to bounding the advantage of $\mathcal{D}$ by the advantage of $\mathcal{B}$ in $\mathsf{Exp}^{\mathsf{nizk}}_{\mathsf{DLog}}(\mathcal{B})$ in the same manner as Claim A.0.1. □

Claim A.0.3. $\mathsf{Adv}^{3}_{\mathcal{D}}(\lambda) < \mathsf{negl}(\lambda)$.

Proof. The cases of $\mathcal{H}_2$ and $\mathcal{H}_3$ differ only in that the A_i are sampled uniformly from $\mathbb{G}$. In order to upper bound $\mathsf{Adv}^{3}_{\mathcal{D}}(\lambda)$, we perform a sequence of ℓ hybrid steps $\mathcal{H}_{3,i}$. On step i we replace $(A_{\ell-i+1}, \ldots, A_\ell) = (a^z_{\ell-i+1}, \ldots, a^z_\ell)$ with $(A_{\ell-i+1} \leftarrow\!\$\, \mathbb{G}, \ldots, A_\ell \leftarrow\!\$\, \mathbb{G})$ (if $\sigma \neq \mathsf{id}$, then we replace each $\{A_j = a^z_{\sigma(j)}\}_{j \geq \ell-i}$). With this notation, $\mathcal{H}_{3,0} = \mathcal{H}_2$ and $\mathcal{H}_{3,\ell} = \mathcal{H}_3$. Without loss of generality, we analyse the distance between $\mathcal{H}_{3,t}$ and $\mathcal{H}_{3,t+1}$. We can bound the advantage of an adversary distinguishing these two games, by constructing a PPT adversary $\mathcal{B}$ that sees the following challenge $(g, g^x, g_1, \ldots, g_\ell, G_1, \ldots, G_\ell)$ where the g_i are random group elements, $G_1, \ldots, G_{n-t-1}$ are set as $g^x_1, \ldots, g^x_{n-t-1}$, $G_{n-t+1}, \ldots, G_\ell$ are random group elements, and G_{n-t} is either g^x_{n-t} or a random group element. $\mathcal{B}$ is then constructed as a simulator of $\mathcal{H}_2$, that also replaces z with x and the $(A_i)_{i=1}^{\ell}$ with $(G_i)_{i=1}^{\ell}$ from the given challenge, and returns b' as returned by $\mathcal{A}$. By application of a hybrid argument over this challenge, given that g^x_i is distributed uniformly in $\mathbb{G}$, then we can bound the advantage of $\mathcal{B}$ by a negligible function. Therefore, if $\ell \in O(\mathsf{poly}(\lambda))$, then $\mathsf{Adv}^{3}_{\mathcal{D}}(\lambda) < \mathsf{negl}(\lambda)$. □

Claim A.0.4. $\mathsf{Adv}^{4}_{\mathcal{D}}(\lambda) = 0$

Proof. At this point, z is only being used to define $Z = g^z$. Since $z \sim \mathbb{F}(\mathbb{G})$ and $\mathbb{G}$ is a prime order group, then $Z \sim U(\mathbb{G})$. Therefore replacing g^z with a uniformly sampled group element is indistinguishable. □

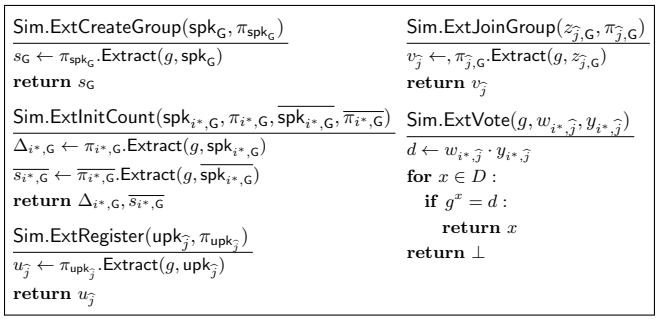

Figure 11. Simulator algorithms extracting adversarial secrets and inputs.

After proving Claim A.0.4, the algorithms Γ.Eval and Γ.SEval, as well as Γ.Gen and Γ.SGen, are identical. Therefore, the cases of $b = 0$ and $b = 1$ are computationally indistinguishable in $\mathsf{Exp}^{\mathsf{ve}}_{\Gamma}(\mathcal{A}, \ell, \lambda)$, and as such $\mathsf{Adv}^{\mathsf{ve}}_{\Gamma,\mathcal{A}}(\lambda, \ell) \leq \mathsf{negl}(\lambda)$. Thus, Lemma 3.2 holds.

Appendix B. Protocol Security Proofs

We now argue the security of the protocol against adversaries that corrupt either $\{\mathsf{G}, \mathcal{C}|_\mathsf{G} \subset \mathcal{V}|_\mathsf{G}, \mathsf{U}_{i^*}\}$ or S. In the following, we construct a simulator that provides an indistinguishable protocol view for such adversaries, using the security model and framework discussed in Section 5. These simulator will simulate each of the algorithms defined in fig. 9 that impact the view of the given adversarial entity.

Notation. For a real-world algorithm Alg run by entity B, the simulated version $\mathsf{Sim}_\mathsf{B}.\mathsf{SAlg}$ produces an indistinguishable view for some adversarial entity $\mathcal{A}_\mathsf{C}$, where the view of C in the real protocol is impacted by the results of Alg. In each case, we write $\mathsf{Adv}^{\mathsf{salg}}_{\mathcal{B}}(\lambda)$ to denote the advantage of some PPT algorithm $\mathcal{D}$ attempting to distinguish between the real and simulated variants.

B.1. Simulation Extraction Algorithms

In fig. 11, we define mechanisms that extract adversarial secrets throughout the protocol, depending on the corrupted entity that is being interacted with. These procedures rely on extracting secret exponents from DLog-like knowledge-extractable NIZK proofs. In Lemma B.1 we prove that each of the DLog extraction algorithms in fig. 11 is PPT, with all but negligible correctness error.

Lemma B.1 (DLog extraction functions)**.** *The* Sim *algorithms* (ExtCreateGroup, ExtInitCount, ExtRegister, ExtJoinGroup) *are PPT with negligible correctness error, assuming the non-interactive knowledge soundness of* DLog.

Proof. In each case, the same logic applies, so we will consider the case of ExtJoinGroup without loss of generality. The simulator receives $(z_{\hat{j},\mathsf{G}} = g^{-s_\mathsf{G} v_{\hat{j}}}, \pi_{\hat{j},\mathsf{G}})$ when $\mathcal{A}_{\mathsf{U}_{\hat{j}}}$ runs JoinGroup to join G. As long as the proof verifies ($\mathsf{DLog.Verify}(z_{\hat{j},\mathsf{G}}, \pi_{\hat{j},\mathsf{G}})$) then, by the non-interactive knowledge soundness of DLog (Definition 2.4), there is a polynomial-time extraction algorithm Extract that extracts the witness (discrete log) $v_{\hat{j}}$ from the known pair $(g^{s_\mathsf{G}}, (g^{s_\mathsf{G}})^{-v_{\hat{j}}})$ with all but non-negligible error. In particular, it is trivial to construct a winning adversary against $\mathsf{Exp}^{\mathsf{nisnd}}_{\mathsf{DLog}}$, based on an algorithm that does not output correctly with all but negligible probability. □

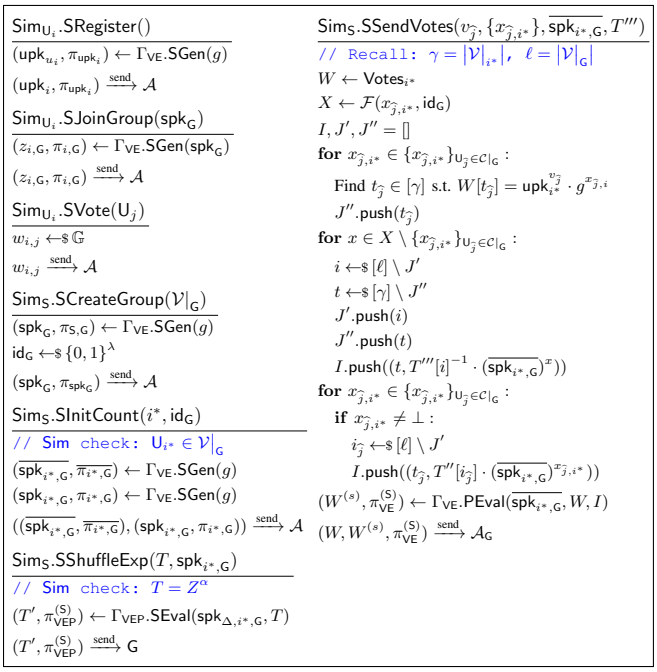

Figure 12. Simulated algorithms for $\mathcal{A} = (\mathcal{A}_\mathsf{G}, \mathcal{A}_{\mathsf{U}_{\hat{j}} \in \mathcal{C}|_\mathsf{G}}, \mathsf{U}_{i^*})$.

Finally, we prove the following short lemma (Lemma B.2) regarding adversarial vote extraction ($\mathsf{ExtVote}(g, w, y)$) returns $x \in D$, such that $w = g^x \cdot y^{-1}$.

Lemma B.2 (Vote extraction)**.** *Assuming that* $|D| = \mathsf{poly}(\lambda)$*,* $\mathsf{Sim.ExtVote}(g, w, y)$ *returns* $x \in D$ *such that* $w = g^x \cdot y^{-1}$ *in PPT, or returns* $\perp$ *otherwise.*

Proof. The proof is trivial based on the construction of ExtVote in fig. 11. Note that the possibility of two different values x and x' satisfying the algorithm is 0, and therefore returning the first satisfying $x \in D$ is correct. □

B.2. Corruption of G, U_{i^*}, and $\mathcal{C}|_\mathsf{G} \subset \mathcal{V}|_\mathsf{G}$

In the $(\mathsf{G}, \mathsf{U}_{i^*}, \mathcal{C}|_\mathsf{G})$ corruption model, we consider a PPT malicious adversary $\mathcal{A} = (\mathcal{A}_\mathsf{G}, \mathcal{A}_{\mathsf{U}_{i^*}}, \mathcal{A}_{\mathsf{U}_{\hat{j}}})$ that corrupts the group admin conducting the tally (G), the target user (U_{i^*}), along with a subset ($\mathcal{C}|_\mathsf{G} \subset \mathcal{V}|_\mathsf{G}$) of group members (who may or may not have voted on the target user). This means that some number of group members (voters) and the facilitation server S remain honest. In order to prove security of the protocol, we demonstrate a simulator Sim that provides a view to $\mathcal{A}$ that is indistinguishable from that

of the real protocol, while only interacting with the ideal functionality $\mathcal{F}$ demonstrated in fig. 8.

Given this corruption model, the simulator must simulate the algorithms shown in fig. 12, since these algorithms all modify or impact the view of the corrupted parties. Our security argument is given and proven in Lemma B.7. First, we prove a number of lemmas that are useful for the eventual security proof. In particular, each lemma proves that the simulated equivalents of functions typically run by either the honest S or an honest U_i (either in the group, or out), are indistinguishable from their real-world counterparts.

Lemma B.3. $\mathsf{Adv}^{\mathsf{x}}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathsf{nizk}}_{\mathcal{B}}(\lambda)$ *for* $\mathsf{x} \in \{\mathsf{SCreateGroup}, \mathsf{SRegister}, \mathsf{SJoinGroup}\}$ *for PPT* $\mathcal{B}$.

Proof. Since the simulation of each of the functions in the set $\{\mathsf{SCreateGroup}, \mathsf{SRegister}, \mathsf{SJoinGroup}\}$ are essentially identical, we give a proof for the case of $\mathsf{SCreateGroup}$, without loss of generality, that covers all of them.

An algorithm $\mathcal{B}$ attempting to violate the non-interactive zero-knowledge property of DLog receives a challenge of the form $(\mathsf{spk}, \pi_{\mathsf{spk}})$, where when $b = 0$, we have that $(s, \mathsf{spk} = g^s, \pi) \leftarrow \Gamma_{\mathsf{VE}}.\mathsf{Gen}(g)$, and when $b = 1$, $(\mathsf{spk}, \pi) \leftarrow \Gamma_{\mathsf{VE}}.\mathsf{SGen}(g)$. In both cases, it is assumed that π verifies correctly. To simulate the distinguishing game in $\mathsf{Exp}^{\mathsf{screategroup}}_{\mathcal{D}}$, $\mathcal{B}$ simply samples $\mathsf{id}_{\mathsf{G}} \leftarrow\$ \{0,1\}^{\lambda}$, and then sends $(\mathsf{id}_{\mathsf{G}}, \mathsf{spk}, \pi)$ to $\mathcal{D}$. If $\mathcal{D}$ can distinguish the two cases with non-negligible advantage, then this transitively implies a non-negligible winning strategy for $\mathcal{B}$. □

The case of $\mathsf{SInitCount}$ is slightly more involved, and the proof argument is given in Lemma B.4, below.

Lemma B.4. $\mathsf{Adv}^{\mathsf{sinitcount}}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{B}'}(\lambda) + 2 \cdot \mathsf{Adv}^{\mathsf{nizk}}_{\mathcal{B}}(\lambda)$ *for any algorithms* $(\mathcal{B}, \mathcal{B}')$.

Proof. In the case of $\mathsf{SInitCount}$, we note that we instantiate a two-step hybrid argument to cover the pair of proven public keys that the adversary receives. In the first step, we replace sampling of $\pi_{i^*,\mathsf{G}} \leftarrow \mathsf{DLog.Prove}(\Delta_{i^*,\mathsf{G}}, g, \mathsf{spk}_{i^*,\mathsf{G}})$ with $(\mathsf{spk}_{i^*,\mathsf{G}}, \pi_{i^*,\mathsf{G}}) \leftarrow \Gamma_{\mathsf{VE}}.\mathsf{SGen}(g)$, by the same argument as in the proof of Lemma B.3. The second step involves noting that the steps that construct $\Delta_{i^*,\mathsf{G}} \leftarrow s_{i^*,\mathsf{G}}/s_{\mathsf{G}}$ and results in the following set of public keys:

$$(g, \overline{\mathsf{spk}_{i^*,\mathsf{G}}} = g^{s_{i^*,\mathsf{G}}}, \mathsf{spk}_{\mathsf{G}} = g^{s_{\mathsf{G}}}, g^{\Delta_{i^*,\mathsf{G}}}),$$

that mirror a Decisional Diffie-Hellman (DDH) challenge (Definition 2.1). Clearly, we can transition to a world where we replace $g^{\Delta_{i^*,\mathsf{G}}}$ with $u \leftarrow\$ \mathbb{G}$ (where $\Delta_{i^*,\mathsf{G}}$ is unknown) via an adversary $\mathcal{B}'$ against DDH.

The final step involves simulating the keypair $(\overline{\mathsf{spk}_{i^*,\mathsf{G}}}, \overline{\pi_{i^*,\mathsf{G}}})$, via the argument used in Lemma B.3. □

We now prove the cases of $\mathsf{ShuffleExp}$, which apply when S and/or U_{i^*} are honest.

Lemma B.5 (Simulation of $\mathsf{SShuffleExp}$)**.** $\mathsf{Adv}^{\mathsf{sshuffleexp}}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathsf{vep}}_{\Gamma_{\mathsf{VEP}},\mathcal{B}}(\lambda)$ *for any PPT algorithm* $\mathcal{B}$.

Proof. Note that the $\mathsf{ShuffleExp}$ is a simple wrapper around the $\Gamma_{\mathsf{VEP}}.\mathsf{eval}$ algorithm, the only difference being that there is a check on the input T (in the case of $\mathsf{S.ShuffleExp}$, or T' in the case of U_{i^*}). These checks are performed by honest voters $\mathsf{U}_j \in \mathcal{V}|_{\mathsf{G}}$, who observe the transcript of communications initiated by G. As noted in Section 5, we make the assumption that the material sent and received by G is written directly to the internal tape of all voters in the group. When G is honest, these checks always pass. In the case where it is not, anyone with knowledge of the set Z can check that $T = Z^{\alpha}$, since $\alpha \leftarrow \mathsf{RO}(Z, \mathsf{upk}_{i^*})$.

Let $\mathcal{B}$ be an adversary attempting to break the security of Γ_{VEP}, then as long as $\mathcal{B}$ is given access to Z, they simply forward the response they get from the VEP experiment to the function caller. Any caller than can distinguish with non-negligible probability translates directly to a non-negligible attack on the underlying VEP scheme. □

Finally, we prove the following lemma, relating to the indistinguishability of Vote and SVote.

Lemma B.6. $\mathsf{Adv}^{\mathsf{svote}}_{\mathcal{D}}(\lambda) = 0$ *for any algorithm* $\mathcal{B}$.

Proof. To prove the valid simulation of the Vote function, we consider an honest U_i voting on a user U_j (who is potentially malicious) but where upk_j is generated honestly (as guaranteed in $\mathsf{U}_j.\mathsf{Register}$). Such a vote is constructed as $w_{i,j} \leftarrow \mathsf{upk}_j^{v_i} \cdot g^{x_{i,j}}$, where $x_{i,j} \in D$. For any $g \in \mathbb{G}$, then g^v is distributed uniformly in $\mathbb{G}$, assuming that $v \leftarrow\$ \mathbb{F}(\mathbb{G})$. Therefore, given that v_i is sampled uniformly, and unknown to U_j, it is trivial to see that $w_{i,j}$ is indistinguishable from being sampled at random from $\mathbb{G}$. □

We now proceed to proving the main security guarantee.

Lemma B.7 (Security against malicious group)**.** *Assume the existence of a secure* VEP *protocol (Definition 3.2), the hardness of solving DDH (Definition 2.1), and a NIZK proof of knowledge system for discrete log relations (Section 2.3). Then the protocol in fig. 9 is secure against corruption of* G, U_{i^*} *and a subset* $\mathcal{C}|_{\mathsf{G}} \subset \mathcal{V}|_{\mathsf{G}}$*, by a malicious adversary* $\mathcal{A}$.

Proof. We detail the hybrid argument below that proves that the real-world protocol and simulation are indistinguishable, where the final step ($\mathcal{H}_{11}$) is equivalent to the simulation. After, follows a series of proven claims that each of the hybrid steps results in two views that are at least computationally infeasible to distinguish.

Hybrid transitions

- $\mathcal{H}_0$: This is the real protocol.
- $\mathcal{H}_1$: For all $\mathsf{U}_{\widehat{j}} \in \mathcal{C}|_{\mathsf{G}}$: runs the extraction algorithm $v_{\widehat{j}} \leftarrow \mathsf{Sim.ExtJoinGroup}(z_{\widehat{j},\mathsf{G}}, \pi_{\widehat{j},\mathsf{G}})$ to extract the voter exponent used by $\mathcal{A}_{\mathsf{U}_{\widehat{j}}}$.
- $\mathcal{H}_2$: For all $\mathsf{U}_{\widehat{j}} \in \mathcal{C}|_{\mathsf{G}}$: runs the extraction algorithm $u_{\widehat{j}} \leftarrow \mathsf{Sim.ExtRegister}(\mathsf{upk}_{\widehat{j}}, \pi_{\mathsf{upk}_{\widehat{j}}})$ to extract the user exponent used by $\mathcal{A}_{\mathsf{U}_{\widehat{j}}}$. Furthermore, runs $u_{i^*} \leftarrow \mathsf{Sim.ExtRegister}(\mathsf{upk}_{i^*}, \pi_{\mathsf{upk}_{i^*}})$ to extract the user exponent used by $\mathcal{A}_{\mathsf{U}_{i^*}}$.
- $\mathcal{H}_3$: For all $\mathsf{U}_{\widehat{j}} \in \mathcal{C}|_{\mathsf{G}}$: runs the extraction algorithm $x_{\widehat{j},i} \leftarrow \mathsf{Sim.ExtVote}(y_{i,\widehat{j}}, \mathsf{spk}_{i,\mathsf{G}}, \mathsf{upk}_i, v_{\widehat{j}})$ to extract the vote

made by $\mathcal{A}_{\mathsf{U}_{\widehat{j}}}$ on any user $\mathsf{U}_i \in \mathcal{U}$ (including U_{i^*}), (or $x_{\widehat{j},i} = \perp$ if $\mathsf{U}_{\widehat{j}}$ didn't vote on U_i).

$\mathcal{H}_4$: Construct the set $\{x_{\widehat{j},i^*}\}_{\mathsf{U}_{\widehat{j}} \in \mathcal{C}|_{\mathsf{G}}}$ of votes cast on U_{i^*} by a corrupted group member, and send it to $\mathcal{F}$, and then receive the total (shuffled) set of votes X made on U_{i^*} (including votes by honest voters).

$\mathcal{H}_5$: Replace calls to S.SendVotes with $\mathsf{Sim}_{\mathsf{S}}$.SSendVotes.

$\mathcal{H}_6$: Replace calls to S.ShuffleExp with $\mathsf{Sim}_{\mathsf{S}}$.SShuffleExp.

$\mathcal{H}_7$: Replace S.InitCount with $\mathsf{Sim}_{\mathsf{S}}$.SInitCount.

$\mathcal{H}_8$: Replace S.CreateGroup with $\mathsf{Sim}_{\mathsf{S}}$.SCreateGroup.

$\mathcal{H}_9$: Replace calls to U_i.Register with $\mathsf{Sim}_{\mathsf{U}_i}$.SRegister.

$\mathcal{H}_{10}$: Replace calls to U_i.JoinGroup with $\mathsf{Sim}_{\mathsf{U}_i}$.SJoinGroup.

$\mathcal{H}_{11}$: Replace calls to U_i.Vote with $\mathsf{Sim}_{\mathsf{U}_i}$.SVote.

In the following, we write $\mathsf{Adv}^{\iota}_{\mathcal{D}}(\lambda)$ to indicate the advantage of a PPT distinguishing algorithm $\mathcal{D}$ in distinguishing between $\mathcal{H}_{\iota-1}$ and $\mathcal{H}_{\iota}$. Furthermore, let $N = |\mathcal{V}|_{\mathsf{G}}|$ and $N_{\mathcal{C}} = |\mathcal{C}|_{\mathsf{G}}|$, be the total number of voters and corrupted voters, respectively. We write $\mathsf{U}_{\widehat{j}}$ when referring to a member of $\mathcal{C}|_{\mathsf{G}}$, where $\widehat{j} \in [N]$.

Claim B.7.1. $\sum_{\iota=1}^{2} \mathsf{Adv}^{\iota}_{\mathcal{D}}(\lambda) < 2(N_{\mathcal{C}}+1) \cdot \mathsf{Adv}^{\mathsf{nisnd}}_{\mathsf{DLog},\mathcal{B}}(\lambda)$ for any PPT algorithm $\mathcal{B}$.

Proof of Claim B.7.1. For any $\mathsf{U}_{\widehat{j}} \in \mathcal{C}|_{\mathsf{G}}$, both the JoinGroup and Register function outputs are sent to all voters in the group and therefore written internally to the immutable transcript seen by honest voters. Thus, crucially, these outputs are observed by the simulator, which allows Sim to run the extraction algorithms on these outputs. More formally, for each $\mathsf{U}_{\widehat{j}} \in \mathcal{C}|_{\mathsf{G}}$, the indistinguishability of both hybrid steps follows from Lemma B.1. Note that the actual view of the adversary does not change in either case — unless soundness of DLog is violated — since the results of the extraction are not exposed. If soundness is violated, the simulation fails further down the line.

In the case of the transition between $\mathcal{H}_1$ and $\mathcal{H}_2$, we must also take into account extracting the secret exponents of U_{i^*} in Register, and in JoinGroup (if they are admitted to the group in the end of the protocol). The explicit extraction steps are exactly the same as in the case of $\mathsf{U}_{\widehat{j}}$. □

Claim B.7.2. $\mathsf{Adv}^{3}_{\mathcal{D}}(\lambda) = 0$.

Proof of Claim B.7.2. Firstly, as in Claim B.7.1, Sim observes all calls to Vote since all votes are sent via the (honest) server. Secondly, since the server has already extracted $v_{\widehat{j}}$ for each $\mathsf{U}_{\widehat{j}} \in \mathcal{C}|_{\mathsf{G}}$, it can The simulator Sim extracts $x_{\widehat{j},i} \in D$ (or $x_{\widehat{j},i} = \perp$, if the vote is malformed) using the argument given in Lemma B.2, for all $\mathsf{U}_i \in \mathcal{U}$. □

Claim B.7.3. $\mathsf{Adv}^{4}_{\mathcal{D}}(\lambda) = 0$.

Proof of Claim B.7.3. Sim constructs the set $\{x_{\widehat{j},i^*}\}$ supplies $x_{\widehat{j},i^*}$ to $\mathcal{F}$, and learns the set X of shuffled plaintext votes via the weak formulation of $\mathcal{F}$ (fig. 8). Note that this happens in the background, and thus does not change the view of the adversary (this will be handled explicitly in Claim B.7.4). □

Claim B.7.4. $\mathsf{Adv}^{5}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathsf{ve}}_{\Gamma_{\mathsf{VE}},\mathcal{B}}(\lambda) + \delta \cdot \mathsf{Adv}^{\mathsf{nisnd}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda)$ for any PPT algorithm $\mathcal{B}$.

Proof of Claim B.7.4. We show that $W^{(s)}$, as computed in Sim.SSendVotes, is computationally indistinguishable from $W^{(s)}$ calculated in $\mathcal{H}_4$. Recall that, in $\mathcal{H}_4$, we compute:

$$(W^{(s)}, \pi^{(s)}_{\mathsf{VE}}) \leftarrow \Gamma_{\mathsf{VE}}.\mathsf{Eval}((\overline{s_{i^*,\mathsf{G}}}, \overline{\mathsf{spk}_{i^*,\mathsf{G}}}), W),$$

where $W = \mathsf{Votes}_{i^*}$. In $\mathcal{H}_5$, we compute:

$$(W^{(s)}, \pi^{(s)}_{\mathsf{VE}}) \leftarrow \Gamma_{\mathsf{VE}}.\mathsf{PEval}(\overline{\mathsf{spk}_{i^*,\mathsf{G}}}, W, I),$$

where $I = \{(t, T''[i] \cdot (\overline{\mathsf{spk}_{i^*,\mathsf{G}}})^x)\}$, for each $x \in X$. We note that, since the simulator has $v_{\widehat{j}}$ and $x_{\widehat{j},i^*}$, they are able to locate the anonymised vote, $w_{\widehat{j},i^*}$, in position $i_{\widehat{j}}$ of W that was made by $\mathcal{A}_{\mathsf{U}_{\widehat{j}}}$ on U_{i^*}. Analysing the set I, we sample random entries (i,t) from W and T''', respectively, and then program VE to output $T'''[t]^{-1} \cdot \overline{\mathsf{spk}_{i^*,\mathsf{G}}}^{x}$ for $W[i]$. The output on $W[i_{\widehat{j}}]$, is programmed also.

Firstly, note that $T'''[t] = g^{-\overline{s_{i^*,\mathsf{G}}} u_{i^*} v_{\sigma\rho(j)}}$ for some $\mathsf{V}_j \in \mathcal{V}|_{\mathsf{G}}$, as long as $\mathcal{A}$ is unable to subvert the protocol. The probability of subversion ($\Pr[\mathsf{Subvert}]$) is calculated based on the actions of $\mathcal{A}_{\mathsf{G}}$.InitExp, the inputs T, T' sent by $\mathcal{A}_{\mathsf{G}}$ to both instances of ShuffleExp, and the response of U_{i^*} in ShuffleExp. In the first case, any group member that has access to Z can check that $\mathcal{A}_{\mathsf{G}}$ calculates $T = Z^{\alpha}$ properly, where $\alpha \leftarrow \mathsf{RO}(Z, \mathsf{upk}_{i^*})$. In the case of the Sim, since all calls to RO are managed by the simulator themselves, this is also checkable. In the cases of sending T and T' we rely on the assumption that all inputs to function calls made by G are written to the internal group transcript, which means that the simulator can check that the group provides the correct inputs to both ShuffleExp invocations. This happens since all calls are routed via the server, which is run internally by Sim. In the case of the response of U_{i^*}, the simulator can check explicitly that $T'' = (T')^{u_{i^*}}$ for some polynomial-time checkable shuffle of T', since it already extracted the target user exponent. Note that, however, if U_{i^*} can violate the soundness of $\pi^{(u)}_{\mathsf{VEP}}$, it could go undetected in $\mathcal{H}_4$. In essence, this would allow U_{i^*} to use a different exponent when exponentiating the votes. Finally, T''' is simply calculated by Sim as $(T'')^{1/o}$. All things considered, $\Pr[\mathsf{Subvert}] = \delta \cdot \mathsf{Adv}^{\mathsf{nisnd}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda)$. Therefore, we note that:

$$\begin{aligned} T'''[t]^{-1} \cdot (\overline{\mathsf{spk}_{i^*,\mathsf{G}}})^x &= g^{\overline{s_{i^*,\mathsf{G}}} u_{i^*} v_{\sigma\rho(j)}} \cdot g^{\overline{s_{i^*,\mathsf{G}}} x} \\ &= (g^{u_{i^*} v_{\sigma\rho(j)} + x})^{\overline{s_{i^*,\mathsf{G}}}} \\ &= \Gamma_{\mathsf{VE}}.\mathsf{Eval}((\overline{s_{i^*,\mathsf{G}}}, \overline{\mathsf{spk}_{i^*,\mathsf{G}}}), W). \end{aligned}$$

Therefore, we conclude that for intersecting votes, we are calculating the corresponding entries in $W^{(s)}$ in $\mathcal{H}_5$ identically. Finally, we must address the ordering of $W^{(s)}$, and the usage of PEval, as opposed to SEval. By assumption, we assume that Votes_{i^*} is received in random order by both U_{i^*} and S. Therefore, by selecting random elements of W to make part of the eventual intersection, the two situations

$\mathsf{Sim}(\mathcal{A}_\mathsf{S})$	$\mathsf{Sim}_\mathsf{G}.\mathsf{STallyVotes}()$
ExtCreateGroup	$X \leftarrow \mathcal{F}(\mathsf{id}_\mathsf{G})$
ExtInitCount	$b_{i^*,\mathsf{G}} \leftarrow \mathsf{Tally}(X)$
ExtRegister	$\mathsf{Sim}_{\mathsf{U}_{i^*}}.\mathsf{SShuffleExp}(T', \mathsf{upk}_{i^*})$
$\mathsf{Sim}_{\mathsf{U}_i}.\mathsf{SRegister}$	// Sim check: $T' \leftarrow$ S.ShuffleExp
$\mathsf{Sim}_{\mathsf{U}_i}.\mathsf{SJoinGroup}$	$(T'', \pi_{\mathsf{VEP}}^{(\mathsf{U}_{i^*})}) \leftarrow \Gamma_{\mathsf{VEP}}.\mathsf{SEval}(\mathsf{upk}_{i^*}, T')$
$\mathsf{Sim}_{\mathsf{U}_i}.\mathsf{SVote}$	$(T'', \pi_{\mathsf{VEP}}^{(\mathsf{U}_{i^*})}) \xrightarrow{\text{send}} \mathcal{A}$

Figure 13. Algorithms to simulate, and additional simulations for $\mathcal{A} = \mathcal{A}_\mathsf{S}$.

are identical. Finally, by Lemma 3.2, the outputs of $\Gamma_{\mathsf{VE}}.\mathsf{eval}$ and $\Gamma_{\mathsf{VE}}.\mathsf{PEval}$ are also indistinguishable. In summary, we conclude that the distinguishing advantage between $\mathcal{H}_4$ and $\mathcal{H}_5$ is bounded by $\mathsf{Adv}^{\mathsf{ve}}_{\Gamma_{\mathsf{VE}},\mathcal{B}}(\lambda) + \delta \cdot \mathsf{Adv}^{\mathsf{nisnd}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda)$. □

Claim B.7.5. $\mathsf{Adv}^{6}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathsf{vep}}_{\Gamma_{\mathsf{VEP}},\mathcal{B}}(\lambda)$

Proof of Claim B.7.5. This step admits a distinguishing advantage bounded by the statement of Lemma B.5. Note that Sim has access to Z, as highlighted in Claim B.7.4, and thus they can carry out the checks necessary to ensure that the VEP protocol is carried out correctlt. Notice that, after this change, the VEP interaction with S is simulated without using the secret exponents of S. □

Claim B.7.6. $\mathsf{Adv}^{7}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{B}'}(\lambda) + 2 \cdot \mathsf{Adv}^{\mathsf{nizk}}_{\mathcal{B}}(\lambda)$ for any PPT algorithms $(\mathcal{B}, \mathcal{B}')$.

Proof of Claim B.7.6. Follows directly from Lemma B.4. After this step, the server's ephemeral exponents are no longer known to the simulator. Note that they are already not used anywhere, due to the simulation of the VE and VEP interactions in $\mathcal{H}_5$ and $\mathcal{H}_6$, respectively. □

Claim B.7.7. $\sum_{\iota=8}^{10} \mathsf{Adv}^{\iota}_{\mathcal{D}}(\lambda) < 3 \cdot \mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{B}}(\lambda)$ for PPT $\mathcal{B}$.

Proof of Claim B.7.7. The distinguishing advantage in each step is bounded by Lemma B.3. Afterwards, all secret exponents of S are unknown. Moreover, all honest users joining the group will send random group elements, and thus their voter tag is unknown. Intuitively, this follows because the voter tags are computed as $\mathsf{spk}_\mathsf{G}^{-v_j}$ for each $\mathsf{V}_j \in \mathsf{G}$. The only other place the tags arise is in the computation of votes by V_j on any $\mathsf{U}_{\hat{j}} \in \mathcal{C}|_\mathsf{G}$. However, since the base of the votes is the generator g, the tags are independently distributed. Therefore, the argument from Lemma B.3 applies. □

Claim B.7.8. $\mathsf{Adv}^{11}_{\mathcal{D}}(\lambda) = 0$ for any algorithm $\mathcal{B}$.

Proof of Claim B.7.8. This follows from Lemma B.6. After this change, the secret exponents for honest voters are no longer exposed at all in the protocol. □

As noted previously, since $\mathcal{H}_{11}$ is identical to the simulation, we can consider the proof of Lemma B.7 complete, given that each of the distinguishing advantages for the hybrid steps is at most $\mathsf{negl}(\lambda)$. □

B.3. Server (S) Corruption

In this corruption model, we consider an adversary $\mathcal{A} = \mathcal{A}_\mathsf{S}$ that corrupts the server. Recall from the discussion in Section 5 that modelling corruptions of the server is much more difficult, and thus we are unable to prove security for $\mathsf{S} + \mathsf{A}$ for $\mathsf{A} \in \{\mathsf{U}_{i^*}, \mathsf{U}_{\hat{j}}, \mathsf{G}\}$ with respect to the base protocol. In Section 8.1, we give an adapted protocol that ensures security in such scenarios. In the base protocol, however, where there is a single group admin, we can still prove security against a malicious server, as long as no further collaboration occurs with any entity associated with the group in question. This means that all internal group members $\mathsf{U}_{\hat{j}} \in \mathcal{V}|_\mathsf{G}$, the group admin itself G, and the target user U_{i^*} remain honest. As before, fig. 13 details the simulated functions. In Lemma B.8, we summarise the security argument for this corruption model.

Lemma B.8. *Assume the existence of a secure* VEP *protocol (Definition 3.2), the hardness of solving DDH (Definition 2.1), and a NIZK proof of knowledge system for discrete log relations (Section 2.3). Then the protocol in fig. 9 is secure against corruption of* S*, by a malicious adversary* $\mathcal{A}$*.*

Proof. The proof follows the hybrid argument below.

Hybrid transitions

- $\mathcal{H}_0$: This is the real protocol.
- $\mathcal{H}_1$: Run $s_\mathsf{G} \leftarrow \mathsf{Sim}.\mathsf{ExtCreateGroup}(\mathsf{spk}_\mathsf{G}, \pi_{\mathsf{spk}_\mathsf{G}})$ to extract the group exponent used by $\mathcal{A}_\mathsf{S}$.
- $\mathcal{H}_2$: Run $\mathsf{Sim}.\mathsf{ExtInitCount}(\mathsf{spk}_{i^*,\mathsf{G}}, \pi_{i^*,\mathsf{G}}, \overline{\mathsf{spk}_{i^*,\mathsf{G}}}, \overline{\pi_{i^*,\mathsf{G}}})$ to extract the secret exponents used by $\mathcal{A}_\mathsf{S}$.
- $\mathcal{H}_3$: Replace the call to G.TallyVotes with calls to $\mathsf{Sim}_\mathsf{G}.\mathsf{STallyVotes}$.
- $\mathcal{H}_4$: Replace $\mathsf{U}_{i^*}.\mathsf{ShuffleExp}$ with $\mathsf{Sim}_{\mathsf{U}_{i^*}}.\mathsf{SShuffleExp}$.
- $\mathcal{H}_5$: Replace $\mathsf{U}_i.\mathsf{Register}$ with $\mathsf{Sim}_{\mathsf{U}_i}.\mathsf{SRegister}$.
- $\mathcal{H}_6$: Replace $\mathsf{U}_i.\mathsf{JoinGroup}$ with $\mathsf{Sim}_{\mathsf{U}_i}.\mathsf{SJoinGroup}$.
- $\mathcal{H}_7$: Replace $\mathsf{U}_i.\mathsf{Vote}$ with $\mathsf{Sim}_{\mathsf{U}_i}.\mathsf{SVote}$.
- $\mathcal{H}_8$: Replace T sent by G in S.ShuffleExp with $T \leftarrow\!\!\$\, \mathbb{G}^\delta$.

We focus on the indistinguishability of $\mathcal{H}_2$ and $\mathcal{H}_3$, which is the only step that differs significantly from those proven in Lemma B.7. Once we reach $\mathcal{H}_8$, this is identical to the ideal-world simulation, and the proof is complete.

Claim B.8.1. $\sum_{\iota=1}^{2} \mathsf{Adv}^{\iota}_{\mathcal{D}}(\lambda) < 3 \cdot \mathsf{Adv}^{\mathsf{nisnd}}_{\mathsf{DLog},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{B}'}(\lambda)$ for any PPT algorithms $(\mathcal{B}, \mathcal{B}')$.

Proof of Claim B.8.1. The indistinguishability of each hybrid step follows directly from Lemma B.1. □

Claim B.8.2. $\mathsf{Adv}^{3}_{\mathcal{D}}(\lambda) < 3 \cdot \mathsf{Adv}^{\mathsf{nisnd}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda)$ for PPT $\mathcal{B}$.

Proof of Claim B.8.2. In $\mathsf{Sim}_\mathsf{G}.\mathsf{STallyVotes}$, the main difference is that the votes considered X are recovered directly from the ideal functionality X, whereas in $\mathcal{H}_3$ they are recovered from the G.IntersectVotes function. An algorithm can distinguish between both hybrids if they can force G.IntersectVotes to output an incorrect set X'. To

perform the analysis of whether such subversion can occur (i.e. $\Pr[\mathsf{Subvert}]$), we consider the structure of the sets $T'', W, W^{(s)}$ provided to the function.

Firstly, in the case of T'', note that the simulator can check that $\mathcal{A}_{\mathsf{S}}$ and $\mathcal{A}_{\mathsf{U}_{i^*}}$ are carrying out the VEP interaction properly, via verification of the proofs $\pi^{(\mathsf{S})}_{\mathsf{VEP}}$, respectively. In particular, by extracting $\Delta_{i^*,\mathsf{G}}$ and u_{i^*} the simulator can check that the returned exponentiated values correspond to a shuffling of the sets $T^{\Delta_{i^*,\mathsf{G}}}$ and $(T^{\Delta_{i^*,\mathsf{G}}})^{u_{i^*}}$, respectively. Therefore, we conclude that the probability of subversion relative to T'' is bounded by $\mathsf{Adv}^{\mathsf{nisnd}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda)$, corresponding the proof issued by $\mathcal{A}_{\mathsf{S}}$ during the VEP interaction.

Secondly, in the case of W, each of the individual honest voters in the group (and thus Sim as well) can check that their vote on U_{i^*} is included. If it is not, they can trigger an abort if it is not included in W received from G.

Thirdly, in the case of $W^{(s)}$, the propensity for $\mathcal{A}$ to subvert the protocol amounts to violating the proof $\pi^{(\mathsf{S})}_{\mathsf{VE}}$ output by Γ_{VE} on W and $\overline{\mathsf{spk}_{i^*,\mathsf{G}}}$. Note that this can be verified and checked by the simulator, who has already extracted the secret exponent $\overline{s_{i^*,\mathsf{G}}}$.

Following this analysis, we conclude that the probability of subversion is calculated as follows:

$$\Pr[\mathsf{Subvert}] < 3 \cdot \mathsf{Adv}^{\mathsf{nisnd}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda).$$

Finally, assuming that no subversion has occurred, then $T''' = \{g^{-\overline{s_{i^*,\mathsf{G}}}u_{i^*}v_j}\}_{\mathsf{U}_j \in \mathcal{V}|_{\mathsf{G}}}$, while $W^{(s)} = \{g^{\overline{s_{i^*,\mathsf{G}}}(v_{j'}u_{i^*}+x_{j',i^*})}\}_{\mathsf{U}'_j \in \mathcal{V}|_{i^*}}$. Let $w_j \in T'''$, when conputing the intersection, we have that:

$$\begin{aligned} w_j \cdot y &= g^{-\overline{s_{i^*,\mathsf{G}}}u_{i^*}v_j} \cdot g^{\overline{s_{i^*,\mathsf{G}}}(v_{j'}u_{i^*}+x_{j',i^*})} \\ &= g^{\overline{s_{i^*,\mathsf{G}}}((v_j - v_{j'})u_{i^*}+x_{j',i^*})}. \end{aligned}$$

Therefore, $w_j \cdot y \cdot g^{-\overline{s_{i^*,\mathsf{G}}}x} = 0$ for any $x \in D$ if $v_j - v_{j'} = 0$ and $x = x_{\widehat{j,i^*}}$, or $x \neq x = x_{\widehat{j,i^*}}$ and $g^{\overline{s_{i^*,\mathsf{G}}}x} = w_j \cdot y$. The latter case occurs with negligible probability, since the value is essentially distributed independently in $\mathbb{G}$. Therefore, this only occurs if both $\mathsf{V}_j \in \mathcal{V}|_{\mathsf{G}}$ and $\mathsf{V}_j \in \mathcal{V}|_{i^*}$.

This condition guarantees that the correct votes are counted, it is then clear that by the fact that each of $(T'', W, W^{(s)})$ are computed correctly, and the fact that $\mathsf{IntersectVotes}$ loops through each element in sequence, this exhaustively counts all such voters satisfying the condition. By the construction of $\mathcal{F}$ therefore, the methods in $\mathcal{H}_4$ and $\mathcal{H}_5$ give equivaleent results except for the occurrence of event $\mathsf{Subvert}$, where $\Pr[\mathsf{Subvert}] < 3 \cdot \mathsf{Adv}^{\mathsf{nisnd}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda)$. □

This concludes the proof of Lemma B.8

Claim B.8.3. $\mathsf{Adv}^{4}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathsf{vep}}_{\Gamma_{\mathsf{VEP}},\mathcal{B}}(\lambda)$

Proof of Claim B.8.3. Noting that all entities in the group are honest, it is trivial to see that this follows immediately from the proof of Lemma B.5. □

Claim B.8.4. $\sum_{\iota=5}^{7} \mathsf{Adv}^{\iota}_{\mathcal{D}}(\lambda) < 2 \cdot \mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{B}}(\lambda)$ for any PPT algorithm $\mathcal{B}$.

Proof of Claim B.8.4. This argument follows the same as those used in the proofs of Claim B.7.7 and Claim B.7.8. □

Claim B.8.5. $\mathsf{Adv}^{8}_{\mathcal{D}}(\lambda) = 0$

Proof. We conclude by showing that the set T sent by G to S can be iteratively (δ times) replaced with fresh samples from $\mathbb{G}$. The reason for this is because $T = Z^{\alpha}$, where $\alpha \leftarrow \mathsf{RO}(Z, \mathsf{upk}_{i^*})$, where $Z = \{z_j^{s_{\mathsf{G}}}\}$ for z_j never revealed to S. As such, α is a uniformly distributed element of $\mathbb{F}(\mathbb{G})$ from the perspective of $\mathcal{A}_{\mathsf{S}}$, and therefore so is the set T. In other words, we can replace it with $T \leftarrow\!\$\, \mathbb{G}^{\delta}$. Notice now that none of the voter tags are ever exposed to the server. □

As noted previously, since $\mathcal{H}_8$ is identical to the simulation. We can consider the proof of Lemma B.8 complete, given that each of the distinguishing advantages for the hybrid steps is at most $\mathsf{negl}(\lambda)$. □