Practical Semi-Open Chat Groups for Secure Messaging Applications

Alex Davidson
LASIGE, Faculdade de Ciências, Universidade de Lisboa

Luiza Soezima
Aarhus University

Fernando Virdia
University of Surrey

Abstract

Secure messaging groups in 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, involving the organisation of protests, activist groups, and internal company dialogues, for instance. Manual administration of who has access to such groups quickly becomes infeasible, in the presence of even hundreds of members.

We construct a practical, privacy-preserving reputation protocol, that automates the approval of new group members based on their reputation amongst the existing membership. We prove security against malicious adversaries in a single-server model, with no further trust assumptions required, while supporting arbitrary reputation calculations even when almost all group members are offline (as is likely). We demonstrate the practicality of the approach experimentally: for groups of size 50 (resp. 500), admitting a user that received 40 (resp. 80) scores requires 1312.2 KiB (resp. 13086.3 KiB) of communication, and 3.4 s (resp. 42.3 s) of single-threaded computation. While our protocol design matches existing secure messaging applications, we believe it can have value in distributed reputation computation beyond this problem setting.

Keywords

secure messaging, e-voting, privacy-preserving reputation

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 grants access to anybody that receives it (*open groups*). However, as groups grow into even hundreds 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 — while groups of thousands of members are considered *already* compromised [1]. In protest/activist organisations [2] and internal company and governmental dialogues, the management of such access is pivotal to ensuring that outsiders/competitors are unable to learn sensitive information. A poignant example of this was in the recent unintended inclusion of journalists in US government Signal groups, that led to global security concerns.

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 [26] (MPC) and e-voting [10] 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 bespoke reputation systems, trust assumptions are common (e.g. via utilisation of 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 [**?**].

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 [**?**]). Along the way, we build novel shuffled exponentiation interactions (Section 5) that may be of independent interest. With these building blocks, we characterise system (Section 2) and security models (Section 3) 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 17.0 s of overall

Proceedings on Privacy Enhancing Technologies 2026(3), 236–257

https://doi.org/10.56553/popets-2026-0080

computation for a group of 200). Note that performance is completely *independent* of the size of the universe of users, and only depends on the size of the group admitting the user U_{i^*}. We discuss remaining limitations and avenues for future work in Section 8. Finally, we note that our protocol is amenable to adaptations that guarantee more advanced functionality and security properties, in the face of stronger collusion models (Appendix A).

Formal contributions.

- An ideal system and security model for constructing practical reputation tallying protocols, and proving security against malicious adversaries.
- A novel and practical verifiable permuted exponentiation (VEP) protocol, based on the hardness of the 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 $\sigma : [n] \mapsto [n]$.
- A practical construction of such a tallying system, based on DDH, said VEP interaction, and zero-knowledge proofs of discrete log relations. We also provide protocol adaptations that handle more complex collusion scenarios, and additional functionality requirements.
- Software implementation[1] and experimental analysis that together demonstrate 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, we desired 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 schemes require group members to be online during the election phase for their vote to be counted. It would also mean that any individual $U_i \in G$ may have to vote multiple times for the same user, for each of the multiple groups that U_i is a member of.

Our approach permits any 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, 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 associated with U_{i^*}, 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 voting group member. We do not discuss a specific measure of reputation, and our methodology thus allows significant flexibility in the choice of reputation function to compute.

To illustrate the overall approach, we describe a simpler variant of our scheme that is secure in an honest-but-curious setting, allowing a single vote to be expressed by U_i on U_{i^*}. Ultimately, we give a protocol that satisfies malicious security (Section 6), and discuss additional functionality adaptations in Appendix A.

Overall design. Let g generate a group $\mathbb{G}$ of prime-order p, where discrete log and decisional Diffie-Hellman (Definition 4.1) are hard problems. In the following description, we will describe the protocol steps, in conjunction with Figures 1 to 5. Each figure defines a set of algorithms that are later defined formally in Section 6.

[1] https://github.com/luizabrs/semi-open-messaging-groups

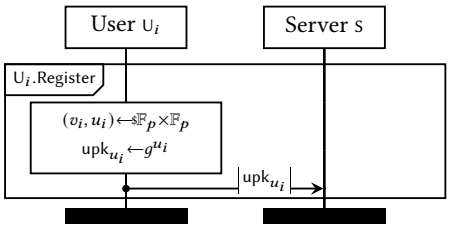

Figure 1: User registration phase.

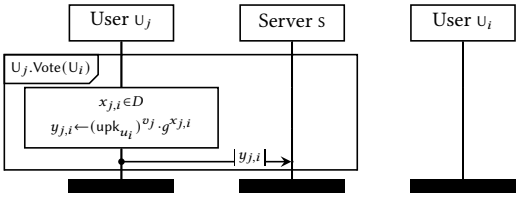

Figure 2: Voting process.

The first phase is user registration (Figure 1). 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.

Any time after registration, user U_j may decide to express a vote on a target user U_i (Figure 2). U_j chooses a score $x_{j,i}$ from a polynomial-sized domain D, and deterministically encrypts it in the exponent as a ballot, $y_{j,i} = (\mathsf{upk}_{u_i})^{v_j} \cdot g^{x_{j,i}}$. This can then be sent via an anonymous channel (e.g. Tor [14] or Oblivious HTTP [**?**]) to the server S. Note that it is important from a security perspective that a user does not hold ballots cast on them, as corrupted users can combine to deanonymise votes sent by the same voter.

At any time, a small number of users can create a "semi-open" group G (Figure 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 $\mathsf{spk}_G = g^{s_G}$, which is provided to every group member (or possibly 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_j to generate a group-specific "intersection tags" $z_{j,G} = (\mathsf{spk}_G)^{-v_j}$, that they share with the other group members. These tags will allow ballots generated by U_j to be detected during reputation tallying of an external user, even if U_j is offline.

Finally, a user U_{i^*} requests to join the group G (Figure 4). We work under the assumption that group members have a way of coordinating, so that all operations performed by them can be considered to happen under a "Group" identity. This can be obtained by having an administrator user that is required to be online to process

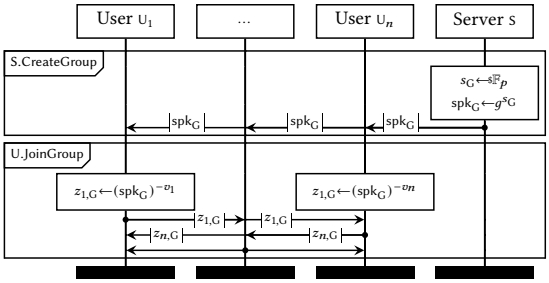

Figure 3: Initial group creation.

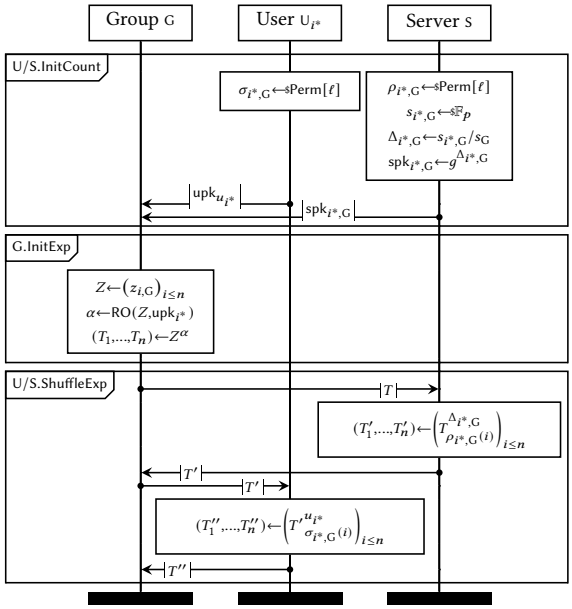

Figure 4: Join request, via shuffled exponentiations.

a join request (though this role can change hands arbitrarily). We assume all group members share the view of the "Group" identity.

Our objective is to be able to intersect the set of ballots cast on U_{i^*}, $W_{i^*} = \{g^{u_{i^*} v_j + x_{j,i^*}}\}_j$, with the intersection tags $z_{i,G} = (spk_G)^{-v_i}$. Mathematically, this requires exponentiation by u_{i^*} on the tags, and by the s_G exponent on the ballots, and then checking for whether the resulting product exists in $\{g^x \mid x \in D\}$. In practice, a few extra steps are required to achieve unlinkability of tags and ballots across join requests. In particular, tags are randomly shuffled before intersection with the ballots, via a novel verifiable permuted exponentiation protocol (Section 5). Furthermore, the server uses an ephemeral group exponent specific to the "join" request by U_{i^*}. After picking random shuffles $\rho_{i^*,G}$, $\sigma_{i^*,G}$ and ephemeral exponent $s_{i^*,G}$ (and corresponding public key $\overline{spk_{i^*,G}}$), the group sends the list of intersection tags, blinded with an exponent α. Server and external user will then shuffle and add their exponents to the tags.

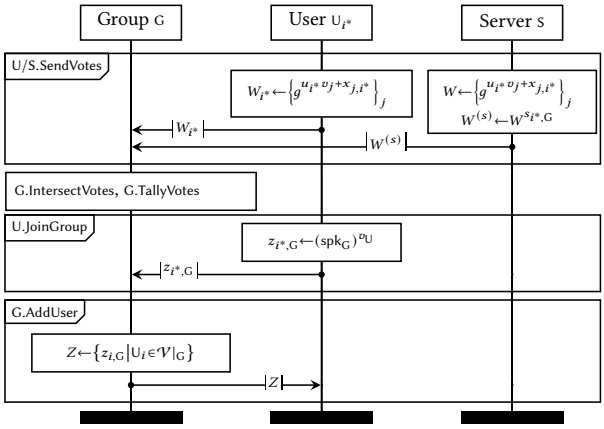

Figure 5: Ballot intersection and tally.

Finally, the server will provide a verifiable copy of the ballots received by user U_{i^*}, with the ephemeral server exponent added (Figure 5). The group can then intersect the ballots and recover the votes, by computing a discrete logarithm of the $g^{s_{i^*,G} \cdot x_{j,i^*}}$ with respect to $\overline{spk_{i^*,G}}$, which is possible in polynomial $O(|D|)$ time due to the polynomial-sized vote domain. The plaintext votes are then tallied to determine whether the external user can be admitted to the group. Any newly admitted user U_{i^*} will provide their intersection tag to the group, and receive those from the other group members.

Tolerating server collusions. The protocol, as described, can be proven secure against any coalition of group users, plus the group admin, plus the target user. In the case of the server, we can only prove security against a non-colluding adversary, which stretches the limits of realism. In Appendix A.1, we discuss a protocol modification that splits the view of the group admin into two distinct roles, only one of which is permitted to be corrupted at any given time. In this model, we can prove security against a colluding server.

Updatable votes. Furthermore, the protocol only allows voting users to submit votes on other users once, while maintaining privacy. Submitting two ballots $g^{u_i v + x_1}$ and $g^{u_i v + x_2}$ on user U_i allows the user to learn $g^{x_1 - x_2}$, which can lead to vote inference attacks. We show in Appendix A.2 how the ballots in our protocol can be adapted to allow submission of up to a bounded number l of unique ballots on a given user. Note that even in our standard protocol, users can update votes by simply revoking their user identity, setting up a new one, and then recasting votes on their peers.

1.2 Wider Discussion and Limitations

We provide some wider discussion around our protocol, along with some known limitations that arise from assumptions that are made during the design process. In Section 6, later highlight additional considerations specifically related to real-world deployments that could inform design further improvements of our protocol. We later detail the full system (Section 2) and security (Section 3) models, along with our modelling assumptions in Section 3.3.

Aiding manual admission processes. As mentioned, the recent incident involving the inclusion of a journalist into a US Government Signal group stresses the utility of automatically obtaining reputation scores for external contacts, relative to a specific messaging group. While we propose the semi-open setting, our mechanism could aid other settings between open and closed, e.g., informing admins of reputation scores during manual admission processes.

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 [**?**], 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 remove the need for ring or group signature schemes, otherwise popular with reputation systems. While such schemes provide useful interfaces, they require the signing party to have access to all public keys in the ring. As we aim to enable voting before group creation, to reduce the burden on the voting party, this would mean holding every public key in the messaging system. With Signal, WhatsApp, and other apps having hundreds of millions of active users, this would not be realistic.

Intrinsic assumptions and limitations. A practical solution to the problem requires the ability of voting once, and then having the resulting ballot count in multiple intersections, and while group members are offline. Group members should not be required to come online to manually vote on an external user, when they want to join the semi-open group. The base protocol in Section 6 satisfies these constraints in the presence of malicious adversaries, by having parties prove they performed their steps correctly, while checking that other parties also performed theirs correctly. These practicality constraints imply some limitations on the resulting protocol.

First, allowing intersections while (some) group members are offline means that these users cannot 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 for the security needs of the group. This is a similar intrinsic risk in automating any group admission process.

Second, it is possible that an external user to the group has not accumulated enough ballots on themselves, and a score cannot meaningfully be computed because the intersection is therefore empty. This problem of having no history as a user is intrinsic to any reputation system, not just our design. The safe approach here would be to fail-closed: notify the group admin(s) that the external user does not have a computable reputation, and delegate the admission decision to them, akin to the closed group setting. 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 would include 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 hands 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 allows her to express a more fine-grained opinion on Bob as a person. However, this may be overkill 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 Ethical and Deployment Considerations

Malformed ballots. We consider two types of invalid input to the reputation system. On the one hand, a malicious user could submit a group element that was not constructed as a valid vote. In this case, the likelihood of the ballot being intersected during $\mathcal{G}$.IntersectVotes is exponentially small due to the size of the group. On the other hand, a user could submit ballots for people with whom they did not actually interact. A possible way to address this issue could involve requiring proofs that voters had engaged meaningfully in some way with the individual that they are voting on. For example, to capture that a meaningful in-app conversation had occurred one could require a zero-knowledge proof of key ratcheting in Signal for a specific number of bidirectional key ratchets. An alternative approach could be using something like rate-limited Privacy Pass [**?**] to limit the number of votes submitted within some period of time, without sacrificing anonymity.

Multiple and lost devices. While our exposition is limited to a basic interface where every user is implicitly assumed to use a single device, in practice most popular messaging platforms support multiple user devices. We foresee two possible avenues for synchronising user identities and key material related to the reputation system across devices. On the one hand, one could construct a key material distribution system similar to that in Signal, where every device has its own identity. Users would then vote on the identities of every known device of another user. Alternatively, one could assume all devices share a single identity, and every new device added by a user could receive the reputation-system key material from the others. This would involve receiving it as an encrypted message within a "group chat", where the members are the various

devices owned by the user. Either option would require careful design in order to achieve robustness, and we believe the problem would deserve an extensive treatment in follow-up work.

Social misuse. The encoding of social dynamics into computerised systems will always carry a risk of enabling exclusionary practices against certain users. In the specific case of reputation systems used for access control, one such risk includes a form of automated online ostracism. For example, if a large enough number of members of an online community voted negatively on a person P, this could result in P being unable to join other groups where the same members are embedded. This can be devastating for community building when the reputation system is *global*, meaning negative votes prevent a target user from joining any messaging groups. We believe this makes systems such as AnonRep [**?**] or PRSONA [**?**] unsuitable for the semi-open group messaging setting.

To reduce this risk, our system is designed from the ground-up as a *local* reputation mechanism, where scores are only meaningful in the context of a specific group of people. Of course, having a negative reputation within a large enough group of members of a specific "community" could make joining groups within the same community circle harder; mimicking, to some extent, the equivalent real-life dynamic. Yet, we emphasize two countermeasures to reduce the risk of such ostracism: default app behaviour for users with negative reputation, and new identity creation. First, our protocol can be nudged in various directions by defining the default behaviour when a user does not have a high-enough group reputation score to be automatically admitted. For example, to avoid unfair negative scoring within a community, external users that are not automatically admitted can be added to a waiting list that is administrated manually. This would essentially downgrade semi-open groups to closed groups, without causing further irreversible damage to user experience. Second, just as with messaging systems such as Signal where new user identities can be relatively easily created, users with an excessively negative reputation score could just abandon their old identity and create a new one. While this process is somewhat burdening, it would simply result in zeroing the reputation scores. Most likely, this would mean that the user in question would again default to being added to waiting lists. Ultimately, though, we believe our localised approach improves upon prior reputation schemes (e.g. [**? ? ?**]) exactly by reducing the risk of ostracism, on top of reducing trust dynamics necessary for instantiating the system via novel cryptographic tooling.

1.4 Related Work

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 [**?**], ARM4FS [**?**], and pRate [23]. 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 [**?**] 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 [**?**] 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 [**?**] 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. Notably, the recently proposed Sandi protocol [**?**] builds a system where users can *downvote* other identities or actions. Aside, from only permitting negative scores, their work only permits a global reputation scoring system. More crucially, the protocol relies on differential privacy (DP) guarantees for hiding such downvotes, which may permit some leakage over time as privacy budgets are expended. Our protocol focuses on general scoring systems, which hide the value of votes beyond what is revealed during vote counting events.

Content Moderation. In content moderation for E2EE [**?**], enforcing the privacy of group-members and reporters is known to be hard, and the use of decentralized properties makes security guarantees even harder, especially when the scheme relies on users reputation. As such, having moderation implies in authentication and traceability, and therefore creating a delicate balance between accountability and client privacy. In bidirectional centralized schemes (where both voter and votee evaluate each other), there exists works such as [**? ?**] that provide discussions on anonymity and likability inside reputation schemes. In fact, the use of centralization is a continual point of discussion [22], where having one single failure point may introduce robustness and trust issues, Other recent works, such as k-Anonymous Group Signatures [**?**], define a more strict moderation process, in which a user becomes traceable only after producing k misbehaviors. Here, the use of threshold-based reputation complies with anonymity, but still requires a centralized authority (with no-privacy tracing power) and a global state. More recent research on "cryptographic personas" [**?**] together with zk-promises [**?**] provide stateful anonymous credentials. While these constructions permit revocation associated with pseudonyms, they still assume global rather than group-scoped reputation.

Webs-of-Trust. Adjacent to cryptography, but without formal cryptographic guarantees, two popular solutions make use of Certificate Authorities (CA) [24] and Webs-of-Trust (WoT) [**?**]. The former is 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. Within more general dynamic systems of entities, reputation has also been treated within the literature on trust systems, where messaging group formation finds a parallel in Virtual Organizations (VO). Security properties are often derived from economic incentives [**?**], though cryptographic designs do exist [**?**].

Delegated MPC protocols. Our protocol can be seen as a specific instantiation of a delegated MPC protocol. In such protocols, clients provide masked versions of their inputs to a set of delegated (typically more powerful) entities, who compute generalised aggregations of the results. The Prio protocol [**?**] proposes a practical mechanism for performing generic computations, but requires at least two non-colluding servers to fulfill the functionality. Similar trust dynamics are required for instantiating general function secret sharing protocols [7], of which practical realisations are known for calculating simple aggregations over client inputs (e.g. for heavy hitter calculations [5]). In all cases, due to the trust assumptions on multiple servers, none of the protocols can be instantiated in the Signal system model, which we view as a core constraint of semi-open group formation schemes. Delegated approaches also exist for constructing private matching schemes [**?**], that allows generalised computation over the output of the blinded matching process, but also require multiple non-colluding servers to instantiate.

2 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 [11]. As such, we build reputation systems assuming only the existence of participants that are *already* supported by Signal (i.e. no new participants are added by our protocol). 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 can 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 Signal involves building 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 public key. 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 only *once*. Once a vote is cast, it remains permanently. In Section 8, 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. Note that ballots cast on U_j are held by the server, and only revealed (blinded) during reputation calculation.

Semi-open groups. Our second modification introduces 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 manual vetting of join requests is considered infeasible. The desire for such groups is common, especially in activism, resistance, and organising contexts. Our solution uses the aforementioned voting overlay to build a reputation system that monitors entries into such groups. The following paragraph summarises the intended functionality.

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 ballots 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. We also assume 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) allows G admission automatically, or be subject to subsequent manual steps, and likewise for rejecting based on *negative* reputation scores (e.g. $k \leq \tau$). 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 non-admin members in $\mathcal{V}|_{\mathsf{G}}$ may be *offline*, and 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. In this case, U_{i^*} enters the set $\mathcal{V}|_{\mathsf{G}}$, and learns all group parameters that existing group members have access to.[2]

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.

[2]This is important for security: group-specific parameters learned by a malicious U_{i^*} on gaining access, should not allow them to open individual ballots that they hold.

3 Security Model

We now describe the security model for semi-open group admissions protocols. Our approach extends to any reputation system/e-voting scenario, given the system design assumed in Section 2.

3.1 Malicious Security Considerations

For a protocol Π, we consider participants $P_1, \ldots, P_n$ and a malicious PPT adversary $\mathcal{A}$ that can selectively corrupt up to $l < n$ parties. Any corruption of P_i gives $\mathcal{A}$ access to their entire internal state, and will allow $\mathcal{A}$ to pick their inputs during the protocol. Furthermore, $\mathcal{A}$ may deviate from the protocol specification arbitrarily: for example, by using malformed inputs, or triggering aborts arbitrarily. 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 in the standalone model, which we formalise in Theorem 3.1 below, with the aid of a PPT simulator Sim, and an ideal functionality $\mathcal{F}$.

Definition 3.1 (Malicious protocol security). Let $\text{View}_{\mathcal{A},\Pi}(1^\lambda)$ denote the view of any PPT algorithm $\mathcal{A}$ in the real-world invocation of Π, and let $\text{View}_{\mathcal{A},\text{Sim},\mathcal{F}}(1^\lambda)$ denote the view of $\mathcal{A}$ in the ideal-world scenario where Sim simulates Π using $\mathcal{F}$. We consider Π secure if, for all $\mathcal{A}$, the following probability is respected:

$$\left|\Pr\left[\text{View}_{\mathcal{A},\Pi}(1^\lambda)\right] - \Pr\left[\text{View}_{\mathcal{A},\text{Sim},\mathcal{F}}(1^\lambda)\right]\right| < \text{negl}(\lambda).$$

Input extraction and step verification. 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 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 are zero-knowledge proof systems that satisfy simulation-extractability [13] (Theorem 4.5), and the extractability of random oracles (that are managed by Sim). Such mechanisms are also used to ensure that adversarial parties abide by the protocol 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.

3.2 Semi-Open Group Model Description

Ideal functionality. As discussed in Section 3.1, 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 Figure 6. 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.

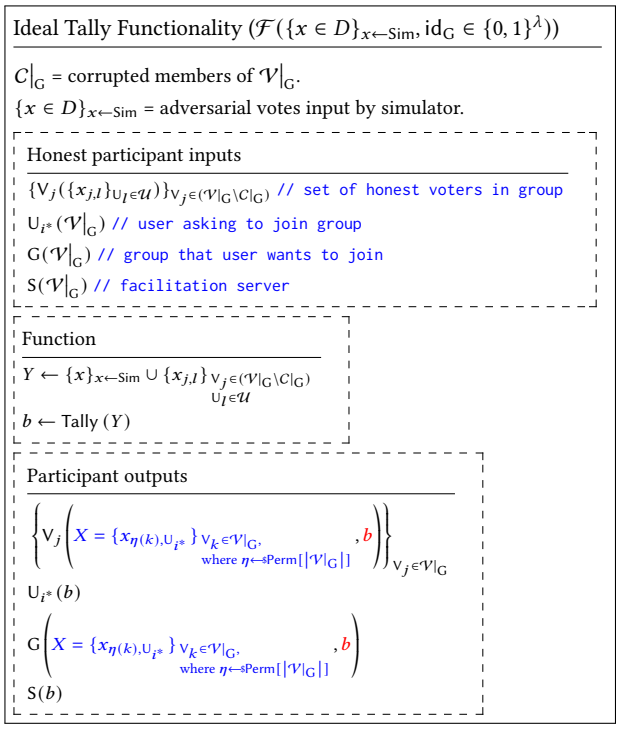

Figure 6: (Weak/Strong) Ideal Functionalities

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.

Multi-session security. We prove security in a single-session setting, consisting of a group, the server, a target user, and a set of group members. A more global view would take into account the full universe of users and groups, and simulate multiple (potentially adaptive) sessions consisting of intersection requests over multiple users, using the ideal functionality above. Such models have been used for analysing security concerning the global view of TLS1.3 connections, for example [15]. We believe our work serves as an important initial step towards achieving security in the multi-session setting, and valuable future work would show that our (or an alternative) proposal satisfies security in this manner.

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. Note that the types of permitted corruptions strengthen or weaken the security

model. For example, proving security against an adversary that only corrupts one of $\{\mathsf{U}_{i^*}, \mathsf{G}, \mathsf{S}, C|_{\mathsf{G}} \subset \mathcal{V}|_{\mathsf{G}}\}$ 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 $\{\mathsf{U}_{i^*}, \mathsf{G}, C|_{\mathsf{G}} \subset \mathcal{V}|_{\mathsf{G}}\}$ 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. 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. In Appendix A.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 prove that Theorem 3.1 holds for each corruption model that is considered. Note that we allow the adversary to abort the protocol arbitrarily.

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 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 $\mathsf{r} \leftarrow \mathsf{RO}(\mathsf{q})$.

3.3 Assumptions

With the system and security models covered, we 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.

Robust internal group transcript. During the protocol defined in Figure 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. In other words, we require that malicious group admins cannot 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 continue for some time afterwards. However, once an honest party comes online and identifies the malicious behaviour, the protocol instance will be abandoned (either during, or afterwards). In the time between malicious behaviour occurring and a triggered abort, no indistinguishability guarantees are given about the real and simulated worlds.

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 $\log_g(c \cdot g^{-v_i u_j}) = \log_g(g^{x_{i,j}})$. By constraining the space of valid votes to a small set D (of polynomial size), this logarithm can be efficiently computed by exhaustive search in $\{g^x \mid x \in D\}$. More advanced "blind intersection" techniques could allow circumventing this limitation, while achieving the stronger ideal functionality in Figure 6. A possible idea would identify 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 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 8).

4 Cryptographic 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 ℓ 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. For an algorithm A, we write $y \leftarrow A(x)$ to denote that A outputs y on input x.

Cryptographic advantage. Let λ be the security parameter, and let $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ denote polynomial and negligible functions, respectively. Let $\mathcal{A}$ be some algorithm attempting to solve an instance of a computational problem XYZ (parameterised by λ). We denote the *advantage* that $\mathcal{A}$ has in solving XYZ by $\mathsf{Adv}^{\mathsf{xyz}}_{\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}\ \text{succeeds}] - \Pr[\mathcal{A}\ \text{fails}]|$. We say that XYZ is *computationally difficult* to solve if, for all PPT $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{xyz}}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

4.1 Cryptographic Groups

We use cyclic (multiplicative) groups $\mathbb{G} = \mathbb{G}(\lambda)$, where the order of the group $\log(q) = \mathsf{poly}(\lambda)$ is prime. 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 $\log(p) =$

poly(λ). We assume all such groups admit hard instances of the discrete log and decisional Diffie-Hellman (Theorem 4.1) problems.

Definition 4.1 (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, thus $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},\mathcal{B}}(\lambda) < \mathsf{negl}(\lambda)$ for all PPT algorithms $\mathcal{B}$.

4.2 NIZK Proof Systems

We recall standard definitions about non-interactive proof systems.

Definition 4.2 (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 $(w, y) \in \mathcal{R}$, returns a proof $\pi \leftarrow \mathsf{Prove}(w, y)$, $\pi \in \mathcal{PS}$,
- 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.

We now give definitions of completeness, non-interactive zero-knowledge, and simulation-extractability, which make up the required guarantees of the proof systems we require.

Definition 4.3 (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 $(w, y) \in \mathcal{R}$, $\Pr[\mathsf{Verify}(y, \mathsf{Prove}(w, y)) = 1] = 1$.

Definition 4.4 (Non-interactive zero knowledge). Let Π be a non-interactive proof system for some relation $\mathcal{R} \subset \mathcal{X} \times \mathcal{Y}$, 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 $w \in \mathcal{X}$. We denote the advantage of an adversary $\mathcal{A}$ attempting to distinguish such statements by $\mathsf{Adv}^{\mathsf{nizk}}_{\Pi,\mathcal{A},\mathsf{Sim}}(\lambda)$.

Definition 4.5 (Non-interactive simulation-extractability). 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 consider a potentially malicious prover algorithm P^* with oracle access to the algorithm Sim, that produces simulated proof statements $y \in \mathcal{Y}$. Suppose now that P^* outputs a valid proof π^* on a particular statement y^*. Then there is a PPT algorithm $\mathsf{Extract}^{P^*}$ that extracts a valid witness w for y^*, with all but negligible probability. In other words, this implies the following:

$$\Pr\left[(w, y^*) \notin \mathcal{R} \;\middle|\; \begin{matrix}(y^*, \pi^*) \leftarrow P^{*\mathsf{Sim}}(1^\lambda) \\ 1 \leftarrow \mathsf{Verify}(y^*, \pi^*) \\ w \leftarrow \mathsf{Extract}^{P^*}(y^*, \pi^*)\end{matrix}\right] < \mathsf{negl}(\lambda).$$

Note that this definition explicitly allows for the extraction algorithm to extract this witness via rewinding, by making multiple calls to P^*. We denote the advantage of an adversary $\mathcal{A}$ attempting to violate the simulation-extractability of a non-interactive proof system Π by $\mathsf{Adv}^{\mathsf{nise}}_{\Pi,\mathcal{A},\mathsf{Sim}}(\lambda)$.

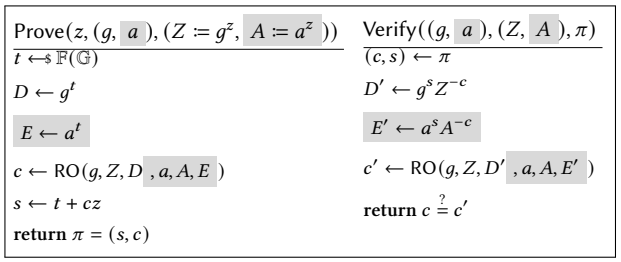

Figure 7: Schnorr-like proofs for DLog [17] and DLEQ [?].

Remark 1. The definition of simulation extraction in Theorem 4.5 differs only from standard knowledge extraction in that P^* has oracle access to Sim. This is vital for extraction of witnesses during protocols where malicious provers may witness simulated proofs (which is the case in the security proof of our eventual construction).

4.3 Discrete-log Proofs

We discuss a series of well-known non-interactive zero-knowledge proof systems for proving relations regarding about statements about discrete logarithm computations in cyclic groups. Each construction utilises the Fiat-Shamir transform in the random oracle model (ROM) for establishing non-interactivity, based on an initial interactive exchange. The guarantees of completeness (Theorem 4.3) and zero-knowledge Theorem 4.4 follow directly from [17]. Meanwhile, the guarantee of simulation-extractability follows directly from the result of [16], which shows that, in the ROM, the Fiat-Shamir transform guarantees non-malleability (and simulation-extractability) of the proof system. This specifically follows when the simulator is allowed to rewind the adversary and the protocol has quasi-unique responses, which is sufficient for proving the eventual security argument of our protocol (see Appendix D). Regarding the requirement for providing quasi-unique responses, we give formal justification that each of the necessary proof systems that we describe below satisfies this restriction in Appendix B.

Proof of Knowledge of Discrete Log (DLog). We use the standard Schnorr proof formulation [25] for constructing NIZKs proofs of knowledge of discrete logarithms, or so-called identification schemes (denoted DLog), as described by Fiat and Shamir [17]. 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 self-containment purposes, we give a formal construction of the proof and verification steps for DLog in Figure 7.

Proof of Discrete Log Equivalence (DLEQ). We give a basic formulation of a DLEQ protocol in Figure 7 (including the additional steps in grey). This formulation is based on the simple formulation given by Chaum and Pedersen [**?**], that has been used in previous verifiable pseudorandom function protocols [21**? ?**]. The fact that it satisfies completeness, and non-interactive zero-knowledge follows immediately from [**?**], while simulation-extractability follows directly from the aforementioned result of [16]. Henceforth, we also write $\pi \leftarrow \mathsf{DLEQ.Prove}(z, (g, \{a_i\}_{i\in[\ell]}), (Z, \{(, A_i)\}_{i\in[\ell]}))$ to

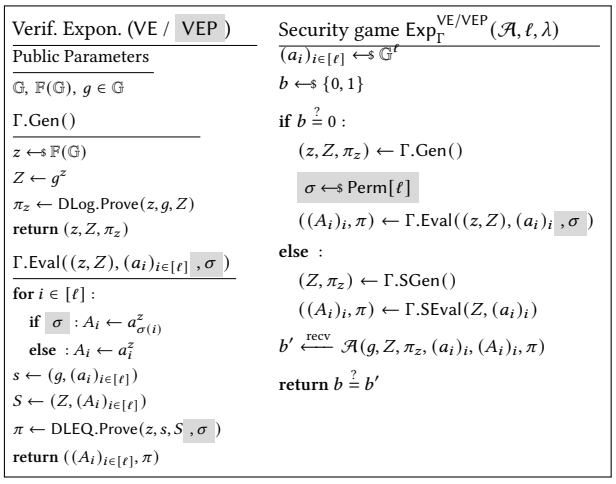

Figure 8: Verifiable exponentiation protocol and security model. Grey backgrounds denote steps used for instantiating the permuted variant, for $\sigma \neq \mathsf{Id}$. We define the simulated variants of the algorithms {SGen, SEval, PEval} in Appendix C.

be ℓ invocations of the standard single-input proof system shown, where π is then a set of ℓ individual DLEQ proofs, and $\pi[i] \leftarrow$ DLEQ.Prove$(z, (g, a_i), (Z, A_i))$. 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, 20].

Shuffled DLEQ. In Section 5, we build a non-interactive zero-knowledge proof of exponentiation-and-shuffle. Given a key pair (g, h), and tuples $(g_1, \ldots, g_n)$ and $(h_1, \ldots, h_n)$, the prover argues that it knows a witness consisting of a secret exponent x and secret permutation σ, such that $h = g^x$, and $h_i = g_{\sigma(i)}^x$. We use the generic compiler of Haines and Müller [19] 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 shuffle-compatible version of the Chaum-Pedersen protocol [**?**] for proving DLEQ, and compile the resulting sigma protocol into a non-interactive proof system using the Fiat-Shamir transform [17]. We can instantiate a standard DLEQ proof system (without shuffle) from this same formulation, by applying an identity permutation. However, for performance reasons, instantiating DLEQ from the Schnorr variant shown in Figure 7 is usually more efficient. We defer the reader to [19] for the full description of the compiler. We adapt the existing notation of DLEQ by writing DLEQ.Prove$(z, (g, \{a_i\}_{i\in[\ell]}), (Z, \{A_{i'}\}_{i'\in[\ell]}), \sigma)$ to indicate the usage of the Shuffled DLEQ protocol, where a correct proof indicates that $a_{\sigma(i)}^z = A_{i'}$ for $i, i' \in [\ell]$.

5 Verifiable Exponentiation and Permutation

Before we build our full protocol, we describe novel protocols for performing 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 a proof that a^z has been computed correctly.

We give a formal construction of two such protocols in Figure 8. The first, VE, that follows this exact framework, and second, VEP, that further 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 4.2) 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 [21**? ?**]). For providing zero-knowledge shuffles, we make use of the general syntax described in [19], but adapted to the case of group exponentiations.

Correctness. Formal correctness for a VE (VEP) protocol follows.

*Definition 5.1 (*VE/VEP *Correctness).* Let Γ be a protocol consisting of algorithms (Gen, Eval), let $\ell = \mathsf{poly}(\lambda)$, and let $\sigma \in \mathsf{Perm}[\ell]$. We say Γ is a correct VEP protocol if the following 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 Figure 8, and Theorem 5.2 captures the associated security condition.

*Definition 5.2 (*VE/VEP *Simulation Security).* Let Γ be a protocol consisting of algorithms (Gen, Eval). We say that Γ is a secure $\mathsf{x} \in \{\mathsf{VE}, \mathsf{VEP}\}$ protocol if, for all PPT algorithms $\mathcal{A}$, then $\mathsf{Adv}_{\Gamma,\mathcal{A}}^{\mathsf{x}}(\lambda, \ell) < \mathsf{negl}(\lambda)$, for all $\ell \in \mathsf{poly}(\lambda)$.

Protocol guarantees. The proofs of correctness and security for our construction, Γ (Figure 8), are given in Appendix C.

5.1 Handling Malicious Inputs

Programmable evaluation. During our eventual 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 randomly. However, the PEval allows providing 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 3 during the proof of Theorem C.2. As a result, as long as the programmed inputs A_j^* are

indistinguishable from random group elements, the programmed evaluation is indistinguishable from fully-simulated evaluation.

Non-random inputs. The protocol in Figure 8 deliberately does not consider *"client-side"* guarantees. 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, 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. Our eventual protocol in Section 6 targets malicious security, meaning 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 3.3 for further discussion on how we handle this in the protocol layer.

6 Protocol Design

We recall notation and conventions. Entity identifiers (and algorithms) are written in camel-case sans-serif font, such as: S (Server), G (group / admin), U_i (i^{th} user in system), U_{i^*} (target user). Sets of entities are written in calligraphic font, such as: $\mathcal{U}$ (all users in ecosystem), $\mathcal{V}|_{\mathsf{G}}$ (voting users belonging to the group), $\mathcal{V}|_i$ (users voting on U_i). For arrays of group elements, we use capital-case (e.g. Z in 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 ballots cast on U_{i^*}, held by S. Additionally, we assume that all NIZK proofs are 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 Figure 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 follow the explanation in Section 1.1, except for the added zero-knowledge proofs that provide 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 protocol, meaning that the outputs of this function will be witnessed by the parties indicated, and thus these outputs must be simulated when these parties are corrupted.

On proof verification, one case to highlight is in S.SendVotes. When G receives this message, each voter in the group (that voted on U_{i^*}) must individually verify their ballot is included (exponentiated) in the set $w^{(s)}$. To do this, they check that one of the DLEQ proofs present in $\pi_{\mathsf{VE}}^{(\mathsf{S})}$ verifies correctly over the sent ballot and a single element in $W^{(s)}$. This check takes $O(\left|W^{(s)}\right|)$ for each individual voter, but can be performed asynchronously when $\mathsf{U}_i \in \mathcal{V}|_{\mathsf{G}}$ comes online. If this does not verify for U_i, it indicates that the server did not send the correct list of ballots, and the intersection event must be aborted. As noted in Section 3, we assume that such aborts can be triggered retrospectively, in the case that U_i is offline during the ballot intersection process.

In summary, we consider the full protocol to be described as the series of messages described in Figures 1 to 5, where each individual algorithm for computing and receiving such messages is described using the implementations described in Figure 9. In particular, this formulation will be adopted when proving correctness and security later, where security will depend on showing indistinguishability of this real protocol execution with a simulation that has access to the *weak* formulation of the ideal functionality described in Figure 6.

Protocol adaptations. Our protocol has two immediate limitations: (i) we can only prove security against a non-colluding server (we discuss the reasons for this more closely in Section 6.1); and (ii) users can only submit a vote on an individual once. We discuss in Appendix A design adaptations to our protocol framework and proof technique to address both of these issues. We further discuss the possibility of tolerating larger vote domains in Appendix A.3, though this would require switching to a *class group* structure (e.g. as used in [9]) and remains relatively open for research.

6.1 Security Guarantees

We consider a target group G, where U_{i^*} is attempting to join. As mentioned in Section 3, we provide security against malicious adversaries corrupting various subsets of entities. In particular, we show that this protocol maintains security against $\mathcal{A} = (\mathcal{A}_{\mathsf{G}}, \mathcal{A}_{\mathsf{U}_{\widehat{j}}}, \mathcal{A}_{\mathsf{U}_{i^*}})$ — 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 Figures 1 to 5 and instantiated via the algorithms in Figure 9 is secure against a malicious adversary that corrupts* either *any subset of entities associated with a given group* G *(Theorem D.6),* or *the facilitation server (Theorem D.7). This follows under the existence of a secure* VEP *protocol (Theorem 5.2), the hardness of DDH, and a NIZK proof of knowledge system for discrete log relations.*

Proof. We refer the reader to Appendix D. □

Remark 2 (Handling server collusions). For handling stronger corruption models (including server collusion), Appendix A.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 $\mathsf{X} \in \{\mathsf{G}, \mathsf{U}_{\widehat{j}}, \mathsf{U}_{i^*}\}$, 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 Appendix A.1.

Intuitive guarantees. Regarding the corruption cases in Theorem 6.1, the first shows that the base protocol is robust to a colluding subset of malicious group members ($\mathsf{G} \cup (C|_{\mathsf{G}} \subset \mathcal{V}|_{\mathsf{G}})$) and the target user ($\mathsf{U}_{i^*}$) 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 S, 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.

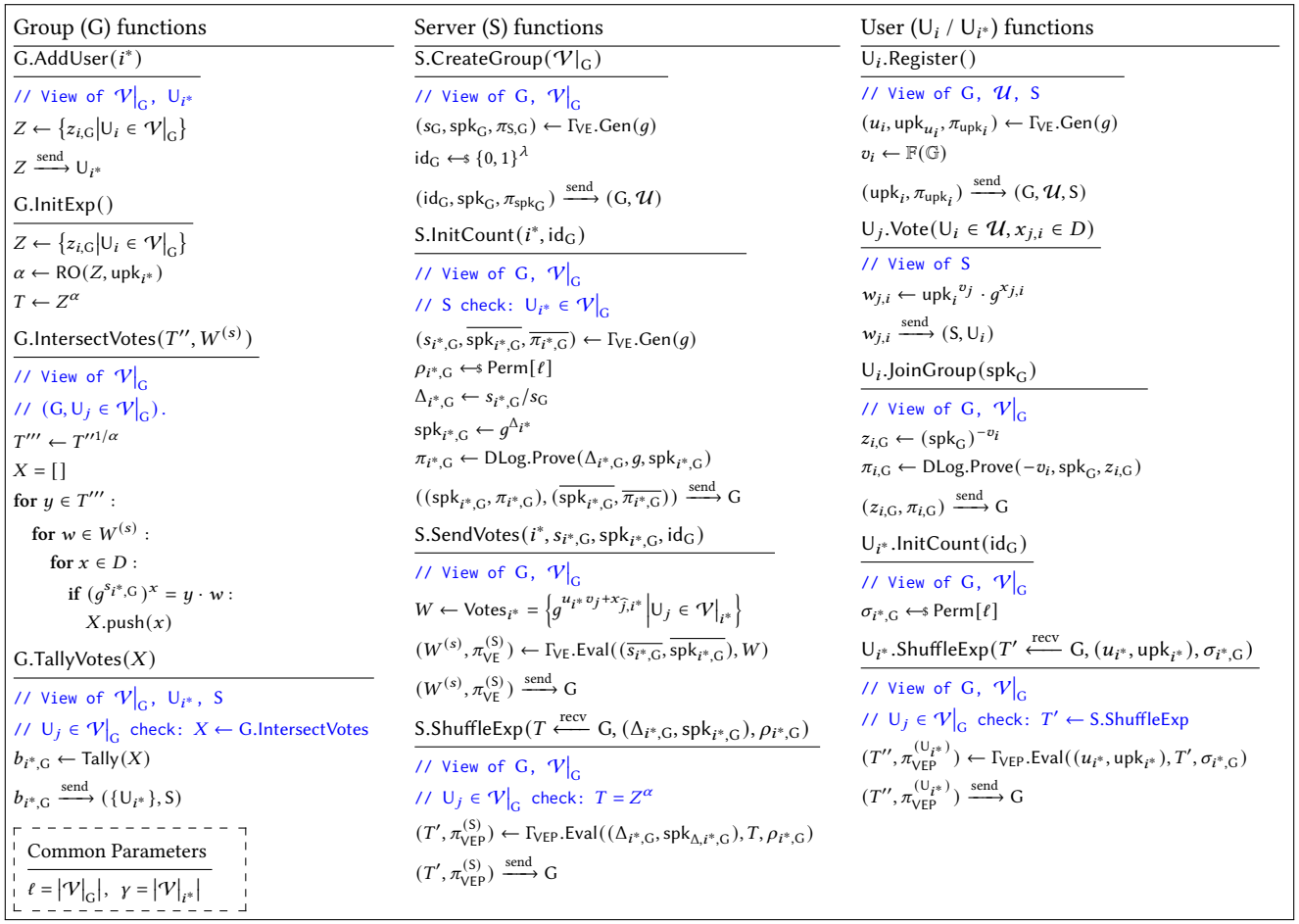

Figure 9: Full list of function implementations used in the message flow described in the base protocol (Section 1.1). For example, G.IntersectVotes **is used in Figure 5.**

Vote confidentiality: Our protocol ensures vote confidentiality meaning that, for any given ballot in the system on U_{i^*}, an adversary cannot discern the value of said vote, apart from via leakage introduced by $\mathcal{F}$ (as in all MPC protocols). Intuitively, this follows from 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 $z_j = g^{-s_G v_j}$. For any $\mathcal{A}$ that does not involve S, then we have an effective DDH relation of the form $(g, g^{1/s_G}, z_j, g^{v_j})$ (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 aforementioned reasons, it is worth noting that ballots themselves are actually unlinkable from the identities of the users casting them. In particular, 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 ballots, the user identities themselves are also hidden (again, accepting the leakage of $\mathcal{F}$ with respect to G).

Tally integrity: We maintain the integrity of the tally computed by G by allowing access to all internal group parameters to entities in $\mathcal{V}|_G$. Since we assume that all communications between the group admin, external user and the server are written to the internal group transcript, internal users can follow the protocol, verify zero-knowledge proofs, and recompute the output of $Tally(X)$ themselves (where X is the set of votes learned in the intersection). 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 4.3, the verifiable exponentiation protocols in Section 5, and the base protocol

Parameters	Phase	Runtime (s) mean	st. dev.	Bandwidth (KiB)
	total	3.4	< 0.1	1312.2
$n = 50$	VE.Eval & check	< 0.1	< 0.1	2.6
$t = 40$	VEP.Eval & check (U)	1.2	< 0.1	653.2
$\|D\| = 10$	VEP.Eval & check (S)	1.2	< 0.1	653.2
	ballot intersection	0.9	< 0.1	1.2
	total	6.6	0.1	2620.1
$n = 100$	VE.Eval & check	< 0.1	< 0.1	2.6
$t = 40$	VEP.Eval & check (U)	2.4	< 0.1	1306.3
$\|D\| = 10$	VEP.Eval & check (S)	2.4	< 0.1	1306.3
	ballot intersection	1.8	< 0.1	1.2
	total	13.3	0.1	5235.7
$n = 200$	VE.Eval & check	< 0.1	< 0.1	2.6
$t = 40$	VEP.Eval & check (U)	4.8	0.1	2612.5
$\|D\| = 10$	VEP.Eval & check (S)	4.8	< 0.1	2612.5
	ballot intersection	3.7	0.1	1.2
	total	17.0	0.1	5239.4
$n = 200$	VE.Eval & check	< 0.1	< 0.1	5.1
$t = 80$	VEP.Eval & check (U)	4.8	< 0.1	2612.5
$\|D\| = 10$	VEP.Eval & check (S)	4.8	0.1	2612.5
	ballot intersection	7.3	< 0.1	2.5
	total	33.2	0.1	13082.6
$n = 500$	VE.Eval & check	< 0.1	< 0.1	2.6
$t = 40$	VEP.Eval & check (U)	12.0	0.1	6531.3
$\|D\| = 10$	VEP.Eval & check (S)	12.0	0.1	6531.3
	ballot intersection	9.1	< 0.1	1.2
	total	42.3	0.1	13086.3
$n = 500$	VE.Eval & check	< 0.1	< 0.1	5.1
$t = 80$	VEP.Eval & check (U)	12.0	0.1	6531.3
$\|D\| = 10$	VEP.Eval & check (S)	12.0	0.1	6531.3
	ballot intersection	18.2	0.1	2.5

Table 1: Costs of base protocol for a group of n users and an external user that received t votes, averaged over 10 runs.

from Section 6 (including all algorithms from Figure 9). To instantiate the protocol, we use the Ristretto255 prime-order group [**?**]. Random oracles within proof systems are implemented using SHA2-256, with the exception of the shuffled DLEQ proof system that uses SHAKE128.[3] 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 open-source here. We use Ristretto and SHA2 implementations from `libsodium` [**?**] (`v1.0.20`), and the SHAKE implementation from the "compact" FIPS202 code in the XKCP library [**?**]. 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 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 [19] 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, by processing loops across multiple threads (since each cycle is independent). 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.[4] All measurements are performed on a Intel(R) Core(TM) Ultra 5 235U CPU on a single core. Overall, a naïve implementation achieves acceptable runtimes for relatively large values of n and t, and shuffled DLEQ soundness error 2^{-128}.

Regarding group sizes, since prior work considers that groups of thousands of users are already compromised (and thus security is already lost) [1], our benchmarks focus only on groups of hundreds. However, noting that ballot intersection uses $O(n \cdot t \cdot |D|)$ group operations, and thus larger groups could be tolerated with similar performance when the vote domain is shrunk accordingly. Further, we stress that no performance comparison was made with other systems, due to the fact that previous constructions consider vastly different modelling assumptions (see Section 1.4 for more details).

8 Future Work and Open Problems

We focus on some open problems associated with our work, that could be the focus of valuable future work. Note that any attempt to address the assumptions and limitations described in Section 1.2 must be careful to not violate the practicality constraints that we enforce in the system model (Section 2).

Currently, we lack an obvious path to building a post-quantum (PQ) version of the protocol. The main hurdle lies in finding practical constructions of 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 [21], none of them appear remotely close to practical deployment [3, 4, 6**? ?**]. Any innovations in this space would design a clearer path to a PQ alternative of our protocol. In terms of the security model, we only satisfy security with respect to the weak formulation of the ideal functionality in Figure 6, 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. We also only prove security in the single-session context, and it would be valuable to consider security in the global, multi-session context. Finally, further investigation into more complex tallies, and how they interact with the overall system, would potentially allow building generic reputation systems that permit more nuanced operations and calculations.

[3] This is for simplicity, as a challenge longer than 256 bits is used.

[4] We do not include costs for single ballots cast on a user before the join request.

Acknowledgments

The authors would like to thank Kevin Gallagher for useful discussions on the topic, Diego F. Aranha for helping identify a stack memory issue with the implementation, and the organizers and attendees at CAW (co-located with IACR Eurocrypt 2024), where a preliminary version of this work was presented. The research of Davidson was funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the project Fully-Homomorphic Encryption from Post-Quantum Code-Based Assumptions, ref. 2024.12595.CMU, DOI https://doi.org/10.54499/2024.12595.CMU, and under the LASIGE Research Unit, ref. UID/00408/2025, DOI https://doi.org/10.54499/UID/00408/2025. The research of Soezima was partially supported by the French National Research Agency in the framework of the "France 2030" program (ANR-11-LABX-0025-01) for the LabEx PERSYVAL while affiliated at Université Grenoble Alpes. The research of Virdia was partly supported by UKRI grant EP/Y02432X/1. Early stages of this work was produced while Davidson and Virdia were affiliated with NOVA LINCS, Universidade NOVA de Lisboa, through the CertiCoLab project, funded by the EEA Grants Bilateral Fund (FBR_OC2_103-CertiCoLab).

References

[1] Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková. 2021. Collective Information Security in Large-Scale Urban Protests: the Case of Hong Kong. In *USENIX Security 2021*. USENIX Association, 3363–3380.

[2] Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková. 2021. Mesh Messaging in Large-Scale Protests: Breaking Bridgefy. In *CT-RSA 2021 (LNCS, Vol. 12704)*. Springer, Cham. https://doi.org/10.1007/978-3-030-75539-3_16

[3] Martin R. Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. 2024. Crypto Dark Matter on the Torus - Oblivious PRFs from Shallow PRFs and TFHE. In *EUROCRYPT 2024, Part VI (LNCS, Vol. 14656)*. Springer, Cham, 447–476. https://doi.org/10.1007/978-3-031-58751-1_16

[4] Martin R. Albrecht, Alex Davidson, Amit Deo, and Nigel P. Smart. 2021. Round-Optimal Verifiable Oblivious Pseudorandom Functions from Ideal Lattices. In *PKC 2021, Part II (LNCS, Vol. 12711)*. Springer, Cham, 261–289. https://doi.org/10.1007/978-3-030-75248-4_10

[5] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2021. Lightweight Techniques for Private Heavy Hitters. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 762–776. https://doi.org/10.1109/SP40001.2021.00048

[6] Dan Boneh, Dmitry Kogan, and Katharine Woo. 2020. Oblivious Pseudorandom Functions from Isogenies. In *ASIACRYPT 2020, Part II (LNCS, Vol. 12492)*. Springer, Cham, 520–550. https://doi.org/10.1007/978-3-030-64834-3_18

[7] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function Secret Sharing. In *EUROCRYPT 2015, Part II (LNCS, Vol. 9057)*. Springer, Berlin, Heidelberg, 337–367. https://doi.org/10.1007/978-3-662-46803-6_12

[8] Johannes Buchmann and Hugh C. Williams. 1988. A Key-Exchange System Based on Imaginary Quadratic Fields. *Journal of Cryptology* 1, 2 (June 1988), 107–118. https://doi.org/10.1007/BF02351719

[9] Guilhem Castagnos and Fabien Laguillaumie. 2015. Linearly Homomorphic Encryption from DDH. In *CT-RSA 2015 (LNCS, Vol. 9048)*. Springer, Cham, 487–505. https://doi.org/10.1007/978-3-319-16715-2_26

[10] David Chaum. 1988. Elections with Unconditionally-Secret Ballots and Disruption Equivalent to Breaking RSA. In *EUROCRYPT'88 (LNCS, Vol. 330)*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45961-8_15

[11] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2020. A Formal Security Analysis of the Signal Messaging Protocol. *Journal of Cryptology* 33, 4 (Oct. 2020), 1914–1983. https://doi.org/10.1007/s00145-020-09360-1

[12] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. 2018. Privacy Pass: Bypassing Internet Challenges Anonymously. *PoPETs* 2018, 3 (July 2018), 164–180. https://doi.org/10.1515/popets-2018-0026

[13] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. 2001. Robust Non-interactive Zero Knowledge. In *CRYPTO 2001 (LNCS, Vol. 2139)*. Springer, Berlin, Heidelberg, 566–598. https://doi.org/10.1007/3-540-44647-8_33

[14] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security 2004*. USENIX Association, 303–320. https://doi.org/10.21236/ada465464

[15] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2021. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol. *Journal of Cryptology* 34, 4 (Oct. 2021), 37. https://doi.org/10.1007/s00145-021-09384-1

[16] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. 2012. On the Non-malleability of the Fiat-Shamir Transform. In *INDOCRYPT 2012 (LNCS, Vol. 7668)*. Springer, Berlin, Heidelberg, 60–79. https://doi.org/10.1007/978-3-642-34931-7_5

[17] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86 (LNCS, Vol. 263)*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-47721-7_12

[18] Stan Gurtler and Ian Goldberg. 2021. SoK: Privacy-Preserving Reputation Systems. *PoPETs* 2021, 1 (Jan. 2021), 107–127. https://doi.org/10.2478/popets-2021-0007

[19] Thomas Haines and Johannes Müller. 2021. A Novel Proof of Shuffle: Exponentially Secure Cut-and-Choose. In *ACISP 21 (LNCS, Vol. 13083)*. Springer, Cham, 293–308. https://doi.org/10.1007/978-3-030-90567-5_15

[20] Ryan Henry and Ian Goldberg. 2013. Batch Proofs of Partial Knowledge. In *ACNS 13International Conference on Applied Cryptography and Network Security (LNCS, Vol. 7954)*. Springer, Berlin, Heidelberg, 502–517. https://doi.org/10.1007/978-3-642-38980-1_32

[21] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. 2014. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. In *ASIACRYPT 2014, Part II (LNCS, Vol. 8874)*. Springer, Berlin, Heidelberg, 233–253. https://doi.org/10.1007/978-3-662-45608-8_13

[22] Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas Zacharias. 2018. On the Security Properties of e-Voting Bulletin Boards. In *SCN 18 (LNCS, Vol. 11035)*. Springer, Cham, 505–523. https://doi.org/10.1007/978-3-319-98113-0_27

[23] Jia Liu and Mark Manulis. 2019. pRate: Anonymous Star Rating with Rating Secrecy. In *ACNS 19International Conference on Applied Cryptography and Network Security (LNCS, Vol. 11464)*. Springer, Cham, 550–570. https://doi.org/10.1007/978-3-030-21568-2_27

[24] Ueli M. Maurer. 1996. Modelling a Public-Key Infrastructure. In *ESORICS'96 (LNCS, Vol. 1146)*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-61770-1_45

[25] Claus-Peter Schnorr. 1990. Efficient Identification and Signatures for Smart Cards. In *CRYPTO'89 (LNCS, Vol. 435)*. Springer, New York, 239–252. https://doi.org/10.1007/0-387-34805-0_22

[26] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *23rd FOCS*. IEEE Computer Society Press. https://doi.org/10.1109/SFCS.1982.38

A Protocol Adaptations

We now present some adaptations of the base protocol. These handle limitations regarding a colluding adversary that corrupts the server and one other entity (Appendix A.1), as well as removing the restriction that any user can only vote once on another user (Appendix A.2), and finally finishing with a discussion of techniques necessary for handling larger vote domains.

A.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

$$\mathsf{X} \in \{\mathsf{G}, \mathsf{U}_{i^*}, \mathcal{C}\big|_{\mathsf{G}} \subset \mathcal{V}\big|_{\mathsf{G}}\}.$$

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.[5] Furthermore, each group member secret-shares their voter tag (in the exponent) amongst both admins, and finally each member is

[5] We allow n admins with one honest, but consider two for simplicity.

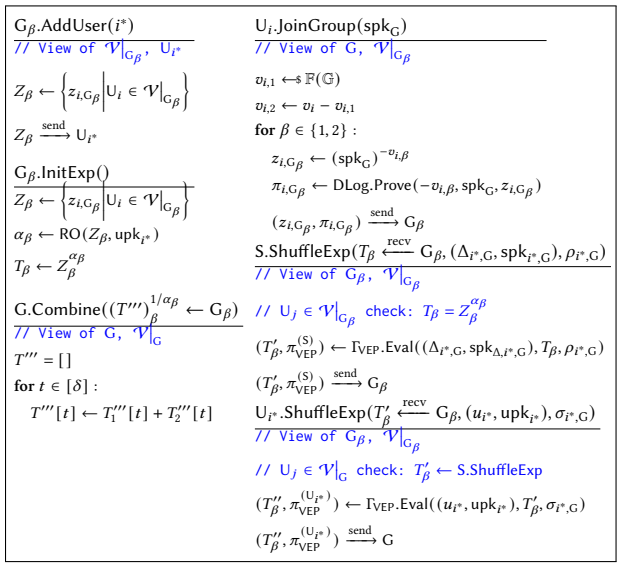

Figure 10: Modified algorithms for adapted (collusion-resistant) protocol.

assigned to the view of only one single admin (i.e. $U_{\tilde{j}}$ belongs to one of $\mathcal{V}|_{G_\beta}$ 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 $\alpha_\beta \leftarrow \text{RO}(Z_\beta, \text{upk}_{i^*})$ 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, $C|_{G_1} \subset \mathcal{V}|_{G_1}$, or U_{i^*}. Security will essentially follow because no entity associated with G_1 (either the admin themselves, the voters assigned to $\mathcal{V}|_{G_1}$, 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 $\mathcal{A} = (\mathcal{A}_S, \mathcal{A}_X)$, 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 $\mathcal{A}_{G_1}$ can choose to send an incorrect T_1''' to G_2, our assumption that at least one entity in both $\mathcal{V}|_{G_\beta}$ is honest prevents this from leading to incorrect results in the protocol.

A.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 $x_{j,i}^{(1)}$ and $x_{j,i}^{(2)}$, then $g^{u_i v_j + x_{j,i}^{(1)}} / g^{u_i v_j + x_{j,i}^{(2)}} = g^{x_{j,i}^{(1)} - x_{j,i}^{(2)}}$, 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, an alternative protocol where votes are constructed as 2-tuples of the form $(g^{u_i v_j}, g^{H(v_j, l) + x_{j,i}^{(l)}})$ (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 $Z_j = \{g^{s_G H(v_j, l)}\}_{l \in [B]}$ 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 $(g^{u_i v_j})$ guarantees that each ballot (and its updates) are linkable, as in the linkable ring signature approach of AnonRep [**?**]. Secondly, ratios of the second element still hide the plaintext vote. However, we note 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 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.

A.3 Larger vote domains

Our protocol only permits voting within polynomially-sized vote domains, since we essentially need to perform discrete logs efficiently to perform the final recovery of vote values from ballots, once the ballot intersection process has completed. We note that class groups of unknown order (as originally introduced by Buchmann and Williams [8]) provide group structures where the discrete logarithm problem is easy to solve, which could be the basis for an alternative construction that would allow specifying votes in exponentially-sized domains. Before getting into challenges that would arise when instantiating such a system, we firstly note that nearly all real-world voting schemes use very small domains (e.g. binary values, or preference rankings from 1 to n for small values of n). However, theoretically speaking, removing this limitation may be interesting in some pathological use-cases.

Our overall goal was to build our system using cryptography that was either standardised, or implementable using standard and well-used cryptographic libraries — such as `libsodium` [**?**], which forms the basis of our eventual implementation (Section 7). Class groups are not typically implemented in such libraries, and we would have to use more experimental code in order to work with such mathematical objects, such as that provided in [9]. On a more

fundamental level, our approach is critically dependent on Schnorr-type zero-knowledge proofs [17] for discrete log arguments, and on verifiable exponentiations that resemble very efficient oblivious pseudorandom function protocols (such as [21]). For both objects, it is clear that security would be violated in groups in which the discrete logarithm problem is easy to solve. Therefore, alternative mechanisms would need to be devised for guaranteeing that the generation of public keys and the exponentiation of certain values was being performed honestly. While it remains unclear if such challenges could be overcome, we reiterate that providing voting capabilities over small domains captures nearly all real-world voting scenarios.

B On Quasi-Uniqueness of Proof Systems

To guarantee the simulation-extractability of the DLog, VE and VEP protocols, we use [16, Thm. 3], which requires the underlying sigma protocols being compiled to have *quasi-unique responses*. Essentially, this requires that for any fixed statement and transcript, a proof system will output a *unique* proof with overwhelming probability. We now argue this is the case for the three protocols.

DLog **protocol.** For proving that public keys are well-formed, we simply prove a DLog relation by using the Fiat-Shamir transform over the Schnorr sigma protocol Σ_{Schnorr}. We prove Σ_{Schnorr} has unique responses and therefore DLog is simulation extractable.

Lemma B.1. *The Schnorr sigma protocol has unique responses over prime-order groups.*

Proof. Observe that given a statement $(g, G) \in \mathbb{G}\times\mathbb{G}$, a transcript $(a, e, z) \in \mathbb{G}\times\mathbb{Z}_q\times\mathbb{Z}_q$ and a second response $z' \in \mathbb{Z}_q$, we can write the check in Verify as $[\![g^z = a \cdot G^e]\!]$. If the check passes for both z and z', over a prime-order group it implies $g^z = g^{z'} \Leftrightarrow z = z'$, meaning that responses are unique. □

DLEQ **protocol.** The proof system necessary for instantiating the VE protocol (Section 5) uses the same underlying sigma protocol used in [12] for batched-DLEQ proofs, proven secure in [**?** , Thm. 3.17]. The protocol has unique responses, and the proof is essentially identical to that for Σ_{Schnorr}.

Lemma B.2. *The batched-DLEQ sigma protocol in [12] has unique responses.*

Proof. In the notation of [12], given a statement $(X, Y, P_1, \ldots, P_m, Q_1, \ldots, Q_m) \in \mathbb{G}^{2+2m}$, a transcript $((A, B), c, s) \in \mathbb{G}^2\times\mathbb{Z}_q\times\mathbb{Z}_q$, and a second response $s' \in \mathbb{Z}_q$, the checks in Verify include $A\cdot Y^{-c} = X^s$. Assuming both s and s' are valid responses, this implies $X^s = X^{s'}$ which over prime-order groups implies $s = s'$. □

Shuffled DLEQ **protocol.** As a core building block of our VEP protocol, we use the sigma protocol for proving DLEQ relations from [19, App. A.5]. We prove that this protocol and the derived shuffled DLEQ protocol have quasi-unique responses, in the sense of [16, Def. 2], as a simpler case of the later lemma for the shuffled protocol.

Lemma B.3. *The Shuffle-compatible DLEQ sigma protocol of [19, App. A.5] provides unique responses.*

Proof. Let $\mathbb{G}$ be a group of prime order q, let (a, e, z) be a protocol transcript where $a = (a_0, a_1) \in \mathbb{G}\times\mathbb{G}$, $e \in \{0, 1\}$, and $z \in \mathbb{G}$, and let $z' \in \mathbb{G}$ be another response, let $x \in \mathbb{Z}_q$ be a witness, and let $y = ((g, g^x), (c_0, c_1))$ be a statement. Then $\mathsf{Verify}(y, (a, e, z)) = 1 \Leftrightarrow (g^{x^e}, c_e)^z = a$ and similarly with z'. It follows that $g^{x^e\cdot z} = g^{x^e\cdot z'}$. Since $\mathbb{Z}_q$ is a field, this implies $z' = z$, meaning the protocol has unique responses. □

We similarly argue that the resulting shuffled DLEQ protocol has quasi-unique responses whenever the elements being shuffled are random. Note that this is sufficient for instantiating our VEP protocol, as discussed in Section 5.1.

Lemma B.4. *The Shuffled DLEQ protocol resulting from the sigma protocol of [19, App. A.5] has quasi-unique responses, when input elements are sampled uniformly from $\mathbb{G}$.*

Proof. We instantiate the Σ_{Shuffle} protocol from [19, Fig. 1], using the shuffle-compatible sigma protocol from [19, App. A.5]. Using the notation [19], as part of our VEP proofs we set all witnesses x^j, for all $j \in [\tau]$, to be the same value x. Given a transcript $((\tilde{a}^j)_{j\in[\tau]}, e, (\tilde{\pi}_z, (\tilde{z}^j)_{j\in[\tau]}))$ and an alternative response $(\tilde{\pi}_{z'}, (\tilde{z'}^j)_{j\in[\tau]})$, the verification condition $g^{x^e\cdot\tilde{z}^j} = g^{\alpha^{\pi_a(j)}}$ implies $\tilde{z}^j = \tilde{z'}^j$ for all $j \in [\tau]$, similarly to Theorem B.3. The further condition $(c_e^{\tilde{\pi}_z(j)})^{\tilde{z}^j} = (c_0^{\pi\circ\pi_a(j)})^{\alpha^{\pi_a(j)}}$, together with $\tilde{z}^j = \tilde{z'}^j$, implies $c_0^{\tilde{\pi}_z(j)} = c_0^{\tilde{\pi}_{z'}(j)}$ for all j. If the $\{c_0^j\}_{j\in[\tau]}$ are distinct, this implies $\tilde{\pi}_z = \tilde{\pi}_{z'}$, which means the response for the given run of Σ_{Shuffle} is unique. Therefore, we can bound $\Pr[\text{non-unique responses}] \le \Pr[\{c_0^j\}_{j\in[\tau]}] \text{ are not distinct}]$. In particular, if the c_0^j are i.i.d. uniformly distributed over $\mathbb{G}$, q is exponentially large and τ polynomially small, then Σ_{Shuffle} has quasi-unique responses. □

Remark 3. We note that Σ_{Shuffle} has to be repeated polynomially many times to reduce the soundness error. Since each run is independent, the overall probability of non-unque responses stays negligibly small.

Remark 4. Theorem B.4 requires assuming that the elements being shuffled are i.i.d. uniformly distributed in $\mathbb{G}$. We point out that in our reputation protocol, the elements being shuffled are exponentiated using an exponent α output by a random oracle evaluation in $\mathcal{G}.\mathsf{InitExp}$ over user tags provided by the members of the messaging group. As long as one of the group members generated the element honestly, the exponent α will be uniformly distributed, and hence the elements being exponentiated.

C Deferred VEP proofs

In this section, we prove the correctness and security of our VEP protocol Γ, as described in Figure 8 in Section 5. The security proof is proven using the simulation detailed in Figure 11.

Lemma C.1 (Correctness of Γ). *The protocol Γ defined in Figure 8 is correct, according to Theorem 5.1.*

Proof. It's clear that $A_{\sigma^{-1}(i)} = a_i^z$ since we have that

$$((A_i)_{i\in[\ell]}, \pi) \leftarrow \Gamma.\mathsf{Eval}((z, Z), (a_i)_{i\in[\ell]}).$$

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 $\sigma = \mathsf{Id}$. □

Simulation

$\Gamma.\mathsf{SGen}()$

$Z \leftarrow\!\$\ \mathbb{G}$
$\pi_z \leftarrow \mathsf{DLog.SProve}(g, Z)$
return (Z, π_z)

$\Gamma.\mathsf{SEval}(Z, (a_i)_{i\in[\ell]})$

return $\Gamma.\mathsf{PEval}(Z, (a_i)_{i\in[\ell]}, \perp)$

$\Gamma.\mathsf{PEval}(Z, (a_i)_{i\in[\ell]}, \{(i_j, A_j^*)\}_{j\in[m]})$

$S_1, S_2 = [\,]$
for $i \in [\ell]$:
  $S_1.\mathsf{push}(a_i)$
  if $\exists\, j \in [m]$ s.t. $i = i_j$
    $S_2.\mathsf{push}(A_j^*)$
  else :
    $A_i \leftarrow\!\$\ \mathbb{G}$
    $S_2.\mathsf{push}(A_i)$
$\pi \leftarrow \mathsf{DLEQ.SProve}((g, S_1), (Z, S_2))$
return (S_2, π)

Figure 11: Simulation for verifiable exponentiation protocol.

Lemma C.2 (Security of Γ). *The protocol* Γ *defined in Figure 8 is a secure* VE (VEP) *protocol, according to Theorem 5.2, for* $\ell \in O(\mathsf{poly}(\lambda))$.

Proof. We show that the case of $b = 0$ in the security model of Figure 8 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 4.3. This is because we eventually prove indistinguishability with a simulator (Figure 11) 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 (Figure 8).
- $\mathcal{H}_1$: 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.

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

Claim 1. $\mathsf{Adv}^1_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathsf{nizk}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda)$
Proof. Let $\mathcal{B}$ be a PPT adversary against the NIZK property of DLEQ ($\mathsf{Exp}^{\mathsf{NIZK}}_{\mathsf{DLEQ}}(\mathcal{B})$). 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 2. $\mathsf{Adv}^2_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathsf{nizk}}_{\mathsf{DLog},\mathcal{B}}(\lambda)$
Proof. Let $\mathcal{B}$ be a PPT adversary against $\mathsf{Exp}^{\mathsf{NIZK}}_{\mathsf{DLog}}(\mathcal{B})$. 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 1. □

Claim 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 $\mathcal{H}_{3,i}$. When addressing step i, we replace $(A_{\ell-i+1}, \ldots, A_\ell) = (a^z_{\ell-i+1}, \ldots, a^z_\ell)$ with the randomly sampled set $(A_{\ell-i+1} \leftarrow\!\$\ \mathbb{G}, \ldots, A_\ell \leftarrow\!\$\ \mathbb{G})$ (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 a distinguisher $\mathcal{A}$ between these two games, by that of a DDH distinguisher $\mathcal{B}$. Specifically, we construct $\mathcal{B}$, which expects in input (g, g^z, g^y, h) and must guess whether $h = g^{z\cdot y}$ or $h \leftarrow\!\$\ \mathbb{G}$, by taking the tuple $(g, g^z, a_1, \ldots a_{\ell-t-1}, g^y, \ldots, a_\ell, A_1, \ldots, A_{\ell-t-1}, h, \ldots, A_\ell)$ and passing it to $\mathcal{A}$. If $\mathcal{A}$ guesses that the tuple was generated in game $\mathcal{H}_{3,t}$, then $\mathcal{B}$ guesses $h = g^{z\cdot y}$, and if $\mathcal{A}$ guesses $\mathcal{H}_{3,t+1}$, $\mathcal{B}$ guesses $h \leftarrow\!\$\ \mathbb{G}$. The illustrative use of g^y and h match the situation in our protocol, where $a_{\ell-t}$ is a random group element,[6] hence $a_{\ell-t} = g^y$, and $A_{\ell-t} \leftarrow a^z_{\ell-t}$. By applying this hybrid ℓ times and by the DDH assumption, if $\ell \in O(\mathsf{poly}(\lambda))$, then $\mathsf{Adv}^3_{\mathcal{D}}(\lambda) < \mathsf{negl}(\lambda)$. □

Claim 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. □

Following Claim 4, the pairs of algorithms $(\Gamma.\mathsf{Gen}, \Gamma.\mathsf{Eval})$ and $(\Gamma.\mathsf{SGen}, \Gamma.\mathsf{SEval})$, are identical. Therefore, the cases $b = 0$ and $b = 1$ are indistinguishable in $\mathsf{Exp}^{\mathsf{VE}}_{\Gamma}(\mathcal{A}, \ell, \lambda)$, and so $\mathsf{Adv}^{\mathsf{VE}}_{\Gamma,\mathcal{A}}(\lambda, \ell) < \mathsf{negl}(\lambda)$. Thus, Theorem C.2 holds.

□

D Protocol Security Proofs

We now argue the security of the protocol against adversaries that corrupt either $\{\mathsf{G}, C|_{\mathsf{G}} \subset \mathcal{V}|_{\mathsf{G}}, \mathsf{U}_{i^*}\}$ 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 3. This simulator will simulate each of the algorithms defined in Figure 9 that impact the view of the given adversarial entity in the protocol defined by the exchanges in Figures 1 to 5.

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

[6] Recall the discussion in Section 5.1, where it is noted that all input elements are sampled uniformly from the group.

$\underline{\mathsf{Sim.ExtCreateGroup}(\mathsf{spk}_\mathsf{G}, \pi_{\mathsf{spk}_\mathsf{G}})}$
$s_\mathsf{G} \leftarrow \mathsf{Extract}((g, \mathsf{spk}_\mathsf{G}), \pi_{\mathsf{spk}_\mathsf{G}})$
return s_G

$\underline{\mathsf{Sim.ExtJoinGroup}(z_{\widehat{j},\mathsf{G}}, \pi_{\widehat{j},\mathsf{G}})}$
$v_{\widehat{j}} \leftarrow \mathsf{Extract}((g, z_{\widehat{j},\mathsf{G}}), \pi_{\widehat{j},\mathsf{G}})$
return $v_{\widehat{j}}$

$\underline{\mathsf{Sim.ExtInitCount}(\mathsf{spk}_{i^*,\mathsf{G}}, \pi_{i^*,\mathsf{G}}, \overline{\mathsf{spk}_{i^*,\mathsf{G}}}, \overline{\pi_{i^*,\mathsf{G}}})}$
$\Delta_{i^*,\mathsf{G}} \leftarrow \mathsf{Extract}((g, \mathsf{spk}_{i^*,\mathsf{G}}), \pi_{i^*,\mathsf{G}})$
$\overline{s_{i^*,\mathsf{G}}} \leftarrow \mathsf{Extract}((g, \overline{\mathsf{spk}_{i^*,\mathsf{G}}}), \overline{\pi_{i^*,\mathsf{G}}})$
return $\Delta_{i^*,\mathsf{G}}, \overline{s_{i^*,\mathsf{G}}}$

$\underline{\mathsf{Sim.ExtVote}(g, w_{i^*,\widehat{j}}, y_{i^*,\widehat{j}})}$
$d \leftarrow w_{i^*,\widehat{j}} \cdot y_{i^*,\widehat{j}}$
for $x \in D$:
  if $g^x = d$:
    return x
return $\perp$

$\underline{\mathsf{Sim.ExtRegister}(\mathsf{upk}_{\widehat{j}}, \pi_{\mathsf{upk}_{\widehat{j}}})}$
$u_{\widehat{j}} \leftarrow \mathsf{Extract}((g, \mathsf{upk}_{\widehat{j}}), \pi_{\mathsf{upk}_{\widehat{j}}})$
return $u_{\widehat{j}}$

Figure 12: Simulator algorithms extracting adversarial secrets and inputs.

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.

Finally, throughout the following, to distinguish between voting users in $\mathcal{V}|_\mathsf{G} \setminus C|_\mathsf{G}$ (honest voters) and voting users in $C|_\mathsf{G}$ (corrupted voters), we will write $\mathsf{V}_j \in \mathcal{V}|_\mathsf{G}$ and $\mathsf{V}_{\widehat{j}} \in C|_\mathsf{G}$. In other words, the subscript $\widehat{j}$ indicates that this corresponds to a corrupted user.

D.1 Simulation Extraction Algorithms

In Figure 12, 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 simulation-extractable NIZK proofs. In Theorem D.1 we prove that each of the DLog extraction algorithms in Figure 12 is PPT, with all but negligible correctness error.

Lemma D.1 (DLog extraction functions). *The* Sim *algorithms* (ExtCreateGroup, ExtInitCount, ExtRegister, ExtJoinGroup) *are PPT with negligible correctness error, assuming the non-interactive simulation extractability 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 $(z_{\widehat{j},\mathsf{G}} = g^{-s_\mathsf{G} v_{\widehat{j}}}, \pi_{\widehat{j},\mathsf{G}})$ when $\mathcal{A}_{\mathsf{U}_{\widehat{j}}}$ runs JoinGroup to join G. As long as the proof verifies ($\mathsf{DLog.Verify}(z_{\widehat{j},\mathsf{G}}, \pi_{\widehat{j},\mathsf{G}})$) then, by the simulation-extractability of DLog (Theorem 4.5), there is a polynomial-time extraction algorithm Extract that extracts the witness (discrete log) $v_{\widehat{j}}$ from the known pair $(g^{s_\mathsf{G}}, (g^{s_\mathsf{G}})^{-v_{\widehat{j}}})$ with all but non-negligible error. In particular, it is trivial to construct a winning adversary against $\mathsf{Exp}^{\mathsf{NISE}}_{\mathsf{DLog}}$, based on an algorithm that does not output correctly with all but negligible probability. □

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

Lemma D.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 Figure 12. 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. □

$\underline{\mathsf{Sim}_{\mathsf{U}_i}.\mathsf{SRegister}()}$
$(\mathsf{upk}_{u_i}, \pi_{\mathsf{upk}_i}) \leftarrow \Gamma_{\mathsf{VE}}.\mathsf{SGen}(g)$
$(\mathsf{upk}_i, \pi_{\mathsf{upk}_i}) \xrightarrow{\text{send}} \mathcal{A}$

$\underline{\mathsf{Sim}_{\mathsf{U}_i}.\mathsf{SJoinGroup}(\mathsf{spk}_\mathsf{G})}$
$(z_{i,\mathsf{G}}, \pi_{i,\mathsf{G}}) \leftarrow \Gamma_{\mathsf{VE}}.\mathsf{SGen}(\mathsf{spk}_\mathsf{G})$
$(z_{i,\mathsf{G}}, \pi_{i,\mathsf{G}}) \xrightarrow{\text{send}} \mathcal{A}$

$\underline{\mathsf{Sim}_\mathsf{S}.\mathsf{SCreateGroup}(\mathcal{V}|_\mathsf{G})}$
$(\mathsf{spk}_\mathsf{G}, \pi_{\mathsf{S},\mathsf{G}}) \leftarrow \Gamma_{\mathsf{VE}}.\mathsf{SGen}(g)$
$\mathsf{id}_\mathsf{G} \hookleftarrow \$ \{0,1\}^\lambda$
$(\mathsf{spk}_\mathsf{G}, \pi_{\mathsf{spk}_\mathsf{G}}) \xrightarrow{\text{send}} \mathcal{A}$

$\underline{\mathsf{Sim}_\mathsf{S}.\mathsf{SInitCount}(i^*, \mathsf{id}_\mathsf{G})}$
// Sim check: $\mathsf{U}_{i^*} \in \mathcal{V}|_\mathsf{G}$
$(\overline{\mathsf{spk}_{i^*,\mathsf{G}}}, \overline{\pi_{i^*,\mathsf{G}}}) \leftarrow \Gamma_{\mathsf{VE}}.\mathsf{SGen}(g)$
$(\mathsf{spk}_{i^*,\mathsf{G}}, \pi_{i^*,\mathsf{G}}) \leftarrow \Gamma_{\mathsf{VE}}.\mathsf{SGen}(g)$
$((\overline{\mathsf{spk}_{i^*,\mathsf{G}}}, \overline{\pi_{i^*,\mathsf{G}}}), (\mathsf{spk}_{i^*,\mathsf{G}}, \pi_{i^*,\mathsf{G}})) \xrightarrow{\text{send}} \mathcal{A}$

$\underline{\mathsf{Sim}_\mathsf{S}.\mathsf{SShuffleExp}(T, \mathsf{spk}_{i^*,\mathsf{G}})}$
// Sim check: $T = Z^a$
$(T', \pi^{(\mathsf{S})}_{\mathsf{VEP}}) \leftarrow \Gamma_{\mathsf{VEP}}.\mathsf{SEval}(\mathsf{spk}_{\Delta,i^*,\mathsf{G}}, T)$
$(T', \pi^{(\mathsf{S})}_{\mathsf{VEP}}) \xrightarrow{\text{send}} \mathsf{G}$

$\underline{\mathsf{Sim}_\mathsf{S}.\mathsf{SSendVotes}(v_{\widehat{j}}, \{x_{\widehat{j},i^*}\}_{\mathsf{U}_{\widehat{j}} \in C|_\mathsf{G}}, \overline{\mathsf{spk}_{i^*,\mathsf{G}}}, T''')}$
// Recall: $\gamma = |\mathcal{V}|_{i^*}|$, $\ell = |\mathcal{V}|_\mathsf{G}|$
$W \leftarrow \mathsf{Votes}_{i^*}$
$X \leftarrow \mathcal{F}(x_{\widehat{j},i^*}, \mathsf{id}_\mathsf{G})$
$I, J', J'' = []$
for $x_{\widehat{j},i^*} \in \{x_{\widehat{j},i^*}\}_{\mathsf{U}_{\widehat{j}} \in C|_\mathsf{G}}$:
  Find $t_{\widehat{j}} \in [\gamma]$ s.t. $W[t_{\widehat{j}}] = \mathsf{upk}_{i^*}^{v_{\widehat{j}}} \cdot g^{x_{\widehat{j},i}}$
  $J''.\mathsf{push}(t_{\widehat{j}})$
for $x \in X \setminus \{x_{\widehat{j},i^*}\}_{\mathsf{U}_{\widehat{j}} \in C|_\mathsf{G}}$:
  $i \hookleftarrow \$ [\ell] \setminus J'$
  $t \hookleftarrow \$ [\gamma] \setminus J''$
  $J'.\mathsf{push}(i)$
  $J''.\mathsf{push}(t)$
  $I.\mathsf{push}((t, T'''[i]^{-1} \cdot (\overline{\mathsf{spk}_{i^*,\mathsf{G}}})^x))$
for $x_{\widehat{j},i^*} \in \{x_{\widehat{j},i^*}\}_{\mathsf{U}_{\widehat{j}} \in C|_\mathsf{G}}$:
  if $x_{\widehat{j},i^*} \neq \perp$:
    $i_{\widehat{j}} \hookleftarrow [\ell] \setminus J'$
    $I.\mathsf{push}((t_{\widehat{j}}, T''[i_{\widehat{j}}] \cdot (\overline{\mathsf{spk}_{i^*,\mathsf{G}}})^{x_{\widehat{j},i^*}}))$
$(W^{(s)}, \pi^{(\mathsf{S})}_{\mathsf{VE}}) \leftarrow \Gamma_{\mathsf{VE}}.\mathsf{PEval}(\overline{\mathsf{spk}_{i^*,\mathsf{G}}}, W, I)$
$(W, W^{(s)}, \pi^{(\mathsf{S})}_{\mathsf{VE}}) \xrightarrow{\text{send}} \mathcal{A}_\mathsf{G}$

Figure 13: Simulated algorithms for $\mathcal{A} = (\mathcal{A}_\mathsf{G}, \mathcal{A}_{\mathsf{U}_{\widehat{j}}} \in C|_\mathsf{G}, \mathsf{U}_{i^*})$**.**

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

In the $(\mathsf{G}, \mathsf{U}_{i^*}, C|_\mathsf{G})$ corruption model, we consider a PPT malicious adversary $\mathcal{A} = (\mathcal{A}_\mathsf{G}, \mathcal{A}_{\mathsf{U}_{i^*}}, \mathcal{A}_{\mathsf{U}_{\widehat{j}}})$ that corrupts the group admin conducting the tally (G), the target user (U_{i^*}), along with a subset ($C|_\mathsf{G} \subset \mathcal{V}|_\mathsf{G}$) 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 Figure 6.

Given this corruption model, the simulator must simulate the algorithms shown in Figure 13, since these algorithms all modify or impact the view of the corrupted parties. Note that simulation of the vote algorithm (Vote) is not necessary in this setting, since ballots are only held by the server, which is played by an honest party. Note that ballots cast by corrupted entities are simply held by the simulator, and used only if needed during the IntersectVotes procedure.

Our security argument is given and proven in Theorem D.6. 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 D.3. $\mathsf{Adv}^{\mathsf{x}}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathsf{NIZK}}_{\mathcal{B}}(\lambda)$ *for PPT* $\mathcal{B}$*, and for* $\mathsf{x} \in \{\mathsf{SCreateGroup}, \mathsf{SRegister}, \mathsf{SJoinGroup}\}$.

Proof. Since the simulation of each of the functions in the set {SCreateGroup, SRegister, SJoinGroup} are essentially identical, we

give a proof for the case of 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 $\mathsf{Exp}_{\mathcal{D}}^{\mathsf{SCreateGroup}}$, $\mathcal{B}$ 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 SInitCount is slightly more involved, and the proof argument is given in Theorem D.4, below.

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

Proof. In the case of 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 Theorem D.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 (Theorem 4.1). 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 Theorem D.3. □

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

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

Proof. Note that the ShuffleExp is a simple wrapper around the Γ_{VEP}.eval algorithm, the only difference being that there is a check on the input T (in the case of 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 3, 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. □

We now proceed to proving the main security guarantee.

Lemma D.6 (Security against malicious group). *Assume the existence of a secure* VEP *protocol (Theorem 5.2), the hardness of solving DDH (Theorem 4.1), and a NIZK proof of knowledge system for discrete log relations (Section 4.3). Then the protocol in Figure 9 is secure against corruption of* G, U_{i^*} *and a subset* $C|_{\mathsf{G}} \subset \mathcal{V}|_{\mathsf{G}}$, *by a malicious adversary* $\mathcal{A}$.

Proof. We detail the hybrid argument below proving that the real-world protocol and simulation are indistinguishable, where the final step ($\mathcal{H}_{10}$) is equivalent to the simulation. After, follows a series of proven claims relating to each of the individual hybrid steps, ensuring (at least) computational indistinguishability.

Hybrid transitions

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

In the following, we write $\mathsf{Adv}_{\mathcal{D}}^{\iota}(\lambda)$ 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 $N_C = |C|_{\mathsf{G}}|$, be the total number of voters and corrupted voters, respectively. We write $\mathsf{U}_{\widehat{j}}$ when referring to a member of $C|_{\mathsf{G}}$, where $j \in [N]$.

Claim 5. $\sum_{\iota=1}^{2} \mathsf{Adv}_{\mathcal{D}}^{\iota}(\lambda) < 2(N_C + 1) \cdot \mathsf{Adv}_{\mathsf{DLog},\mathcal{B}}^{\mathsf{NISE}}(\lambda)$ for any PPT algorithm $\mathcal{B}$.

Proof. [Proof of Claim 5] For any $\mathsf{U}_{\widehat{j}} \in C|_{\mathsf{G}}$, 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 $\mathsf{U}_{\widehat{j}} \in C|_{\mathsf{G}}$, the indistinguishability of both hybrid steps follows from Theorem D.1. Note that the actual view of the adversary does not change in either case — unless simulation-extractability of DLog is violated — since the results of the extraction are not exposed. If this 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 6. $\mathsf{Adv}^{3}_{\mathcal{D}}(\lambda) = 0$.

Proof. [Proof of Claim 6] Firstly, as in Claim 5, 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 $\mathsf{U}_{\widehat{j}} \in C|_{\mathsf{G}}$, 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 Theorem D.2, for all $\mathsf{U}_i \in \mathcal{U}$. □

Claim 7. $\mathsf{Adv}^{4}_{\mathcal{D}}(\lambda) = 0$.

Proof. [Proof of Claim 7] 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 $\mathcal{F}$ (Figure 6). Note that this happens in the background, and thus does not change the view of the adversary (this will be handled explicitly in Claim 8). □

Claim 8. $\mathsf{Adv}^{5}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathsf{VE}}_{\Gamma_{\mathsf{VE}},\mathcal{B}}(\lambda) + \delta \cdot \mathsf{Adv}^{\mathsf{NISE}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda)$ for any PPT algorithm $\mathcal{B}$.

Proof. [Proof of Claim 8] 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. Note that W is never observed by the adversary, and so we are free to sample random entries from the set for simulating votes.

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[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 simulation-extractability 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, $\Pr[\mathsf{Subvert}] = \delta \cdot \mathsf{Adv}^{\mathsf{NISE}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda)$. 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 usage of PEval, as opposed to SEval. Finally, by Theorem C.2, the outputs of Γ_{VE}.eval and Γ_{VE}.PEval are also indistinguishable. In summary, we conclude that the distinguishing advantage between $\mathcal{H}_4$ and $\mathcal{H}_5$ is bounded by $\mathsf{Adv}^{\mathsf{VE}}_{\Gamma_{\mathsf{VE}},\mathcal{B}}(\lambda) + \delta \cdot \mathsf{Adv}^{\mathsf{NISE}}_{\mathsf{DLEQ},\mathcal{B}}(\lambda)$. □

Claim 9. $\mathsf{Adv}^{6}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathsf{VEP}}_{\Gamma_{\mathsf{VEP}},\mathcal{B}}(\lambda)$

Proof. [Proof of Claim 9] This step admits a distinguishing advantage bounded by the statement of Theorem D.5. Note that Sim has access to Z, as highlighted in Claim 8, 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 10. $\mathsf{Adv}^{7}_{\mathcal{D}}(\lambda) < \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},\mathcal{B}'}(\lambda) + 2 \cdot \mathsf{Adv}^{\mathsf{NIZK}}_{\mathcal{B}}(\lambda)$ for any PPT algorithms $(\mathcal{B}, \mathcal{B}')$.

Proof. [Proof of Claim 10] Follows directly from Theorem D.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 11. $\sum_{i=8}^{10} \mathsf{Adv}^{i}_{\mathcal{D}}(\lambda) < 3 \cdot \mathsf{Adv}^{\mathsf{NIZK}}_{\mathcal{B}}(\lambda)$ for PPT $\mathcal{B}$.

Proof. [Proof of Claim 11] The distinguishing advantage in each step is bounded by Theorem D.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 $\mathsf{U}_{\widehat{j}} \in C|_{\mathsf{G}}$. However, since the base of the votes is the generator g, the tags are independently distributed. Therefore, the argument from Theorem D.3 applies. □

As noted previously, since $\mathcal{H}_{10}$ is identical to the simulation, we can consider the proof of Theorem D.6 complete, given that each of the distinguishing advantages for the hybrid steps is at most $\mathsf{negl}(\lambda)$. □

D.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 3 that modelling corruptions of the server is much more difficult, and thus we are unable to prove security for S + A for $\mathsf{A} \in \{\mathsf{U}_{i^*}, \mathsf{U}_{\widehat{j}}, \mathsf{G}\}$ with respect to the base protocol. In Appendix A.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 $\mathsf{U}_{\widehat{j}} \in \mathcal{V}|_{\mathsf{G}}$, the group admin itself G, and the target user U_{i^*} remain honest. As before, Figure 14 details the simulated functions. In Theorem D.7, we summarise the security argument for this corruption model.

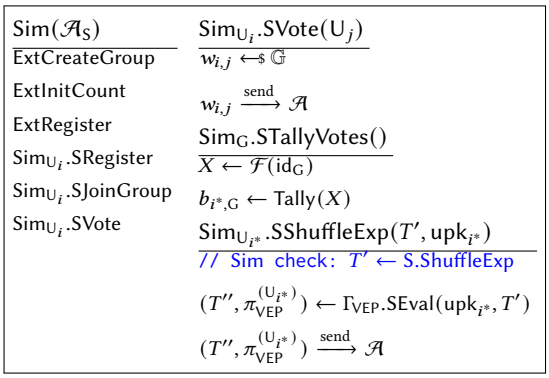

Figure 14: Algorithms to simulate, and additional simulations for $\mathcal{A} = \mathcal{A}_S$.

Lemma D.7. *Assume the existence of a secure* VEP *protocol (Theorem 5.2), the hardness of solving DDH (Theorem 4.1), and a NIZK proof of knowledge system for discrete log relations (Section 4.3). Then the protocol in Figure 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 $s_G \leftarrow$ Sim.ExtCreateGroup(spk$_G$, π_{spk_G}) to extract the group exponent used by $\mathcal{A}_S$.
- $\mathcal{H}_2$: Run Sim.ExtInitCount(spk$_{i^*,G}$, $\pi_{i^*,G}$, $\overline{\text{spk}_{i^*,G}}$, $\overline{\pi_{i^*,G}}$) to extract the secret exponents used by $\mathcal{A}_S$.
- $\mathcal{H}_3$: Replace the call to G.TallyVotes with calls to Sim$_G$.STallyVotes.
- $\mathcal{H}_4$: Replace U$_{i^*}$.ShuffleExp with Sim$_{U_{i^*}}$.SShuffleExp.
- $\mathcal{H}_5$: Replace U$_i$.Register with Sim$_{U_i}$.SRegister.
- $\mathcal{H}_6$: Replace U$_i$.JoinGroup with Sim$_{U_i}$.SJoinGroup.
- $\mathcal{H}_7$: Replace U$_i$.Vote with Sim$_{U_i}$.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 D.6. Once we reach $\mathcal{H}_8$, this is identical to the ideal-world simulation, and the proof is complete.

Claim 12. $\sum_{\iota=1}^{2} \text{Adv}^{\iota}_{\mathcal{D}}(\lambda) < 3 \cdot \text{Adv}^{\text{NISE}}_{\text{DLog},\mathcal{B}}(\lambda) + \text{Adv}^{\text{ddh}}_{\mathbb{G},\mathcal{B}'}(\lambda)$ for any PPT $(\mathcal{B}, \mathcal{B}')$.

Proof. [Proof of Claim 12] Indistinguishability of each step follows from Theorem D.1. □

Claim 13. $\text{Adv}^{3}_{\mathcal{D}}(\lambda) < 3 \cdot \text{Adv}^{\text{NISE}}_{\text{DLEQ},\mathcal{B}}(\lambda)$ for PPT $\mathcal{B}$.

Proof. [Proof of Claim 13] In Sim$_G$.STallyVotes, the main difference is that the votes considered in X are output directly from the ideal functionality X, whereas in $\mathcal{H}_3$ they are output by the G.IntersectVotes function. An algorithm can distinguish between the hybrids if they can force G.IntersectVotes to output an incorrect set X'. To analyse whether such subversion can occur (i.e. Pr[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 $\mathcal{A}_S$ and $\mathcal{A}_{U_{i^*}}$ are carrying out the VEP interaction properly, via verification of the proofs $\pi^{(S)}_{VEP}$, respectively. In particular, by extracting $\Delta_{i^*,G}$ and u_{i^*} the simulator can check that the returned exponentiated values correspond to a shuffling of the sets $T^{\Delta_{i^*,G}}$ and $(T^{\Delta_{i^*,G}})^{u_{i^*}}$, respectively. Therefore, we conclude that the probability of subversion relative to T'' is bounded by $\text{Adv}^{\text{NISE}}_{\text{DLEQ},\mathcal{B}}(\lambda)$, corresponding the proof issued by $\mathcal{A}_S$ 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 $\pi^{(S)}_{VE}$ output by Γ_{VE} on W and $\overline{\text{spk}_{i^*,G}}$. Note that this can be verified and checked by the simulator, who has already extracted the secret exponent $\overline{s_{i^*,G}}$.

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

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

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

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

Therefore, $w_j \cdot y \cdot g^{-\overline{s_{i^*,G}} x} = 0$ 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 $g^{\overline{s_{i^*,G}} x} = w_j \cdot y$. The latter case occurs with negligible probability, since the value is essentially distributed independently in $\mathbb{G}$. Therefore, this only occurs if both $V_j \in \mathcal{V}|_G$ and $V_j \in \mathcal{V}|_{i^*}$.

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 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 equivalent results except for the occurrence of event Subvert, where $\Pr[\text{Subvert}] < 3 \cdot \text{Adv}^{\text{NISE}}_{\text{DLEQ},\mathcal{B}}(\lambda)$. □

Claim 14. $\text{Adv}^{4}_{\mathcal{D}}(\lambda) < \text{Adv}^{\text{VEP}}_{\Gamma_{\text{VEP}},\mathcal{B}}(\lambda)$

Proof. [Proof of Claim 14] Noting that all entities in the group are honest, it is trivial to see that this follows immediately from the proof of Theorem D.5. □

Claim 15. $\sum_{\iota=5}^{6} \text{Adv}^{\iota}_{\mathcal{D}}(\lambda) < 2 \cdot \text{Adv}^{\text{NIZK}}_{\mathcal{B}}(\lambda)$ for any PPT algorithm $\mathcal{B}$.

Proof. [Proof of Claim 15] This argument follows from the proof of Claim 11. □

Claim 16. $\text{Adv}^{\text{SVote}}_{\mathcal{D}}(\lambda) = 0$ 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$_j$. Such a vote is constructed as $w_{i,j} \leftarrow \text{upk}_j^{v_i} \cdot g^{x_{i,j}}$, where $x_{i,j} \in D$. Since upk$_j$ is generated honestly, then g^{v_i} is distributed uniformly in $\mathbb{G}$, since $v_i \leftarrow\$ \mathbb{F}(\mathbb{G})$. As a result it is clear to see that $w_{i,j}$ is indistinguishable from being sampled at random from $\mathbb{G}$. □

Claim 17. $\text{Adv}^{8}_{\mathcal{D}}(\lambda) = 0$

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 $Z = \{z_j^{s_G}\}$ for z_j never revealed to S. As such, α is a uniformly distributed element of $\mathbb{F}(\mathbb{G})$ from the perspective of $\mathcal{A}_S$, 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 Theorem D.7 complete, given that each of the distinguishing advantages for the hybrid steps is at most $\mathsf{negl}(\lambda)$. □