PeerVote: A Voter-to-Voter Internet Voting Protocol with Federated Distributed Key Generation

No Author Given

No Institute Given

Abstract. Internet voting is essential in today's digital democracy, yet the landscape of Internet voting is dominated by centralised servers, costly public blockchains or private networks. This poses a challenge for small organisations looking for a secure, cost-effective tool for democratic decision-making. This paper introduces PeerVote, an Internet voting protocol designed to address these challenges by providing a secure voting mechanism without the need for trusted third parties. PeerVote uses Federated Distributed Key Generation (FDKG) to improve resilience to node unavailability and introduces a flexible trust model, providing small organisations with a viable tool for promoting democratic engagement.

Keywords: Internet Voting · Digital Democracy · Federated Distributed Key Generation · Threshold Cryptography · zkSNARKs · Blockchain

1 Introduction

Voting is a fundamental mechanism for collective decision-making, used in contexts ranging from student associations, non-governmental organisations and corporate boardrooms to national presidential elections and global online polls.

Voting methods are diverse and include traditional paper-based voting, mail-in ballots, electronic systems such as direct recording electronic (DRE) machines, and Internet voting [**?**].

As Vitalik Buterin has noted [**?**], every voting system faces a trilemma, requiring a choice between two of three critical attributes:

– **Democratic**: Ensuring fair and accessible participation for all eligible voters.
– **Secure**: Ensuring integrity, transparency, privacy and resilience to potential threats.
– **Efficient**: Achieving simplicity, speed and cost-effectiveness in the voting process.

In traditional political elections, the emphasis on security and democracy often leads to a compromise on efficiency. Conversely, social media voting prioritises democracy and efficiency at the expense of security. The market system represents an efficient and secure decision-making model, where consumer choices influence corporate power, but it lacks democratic inclusiveness, making it unsuitable for decision-making on public goods.

The inherent inefficiencies of traditional voting methods result in significant costs and infrequent election cycles, typically ranging from once a year to once every six years [?].

Internet voting (i-voting) is emerging as a seemingly ideal solution, especially when online banking is so widespread these days. Especially during events such as the COVID-19 pandemic, i-voting presents itself as a conventional, cost-effective, fast and secure alternative. Its potential to increase voter turnout, increase the frequency of elections and facilitate different democratic models such as direct democracy, liquid democracy and alternative voting systems is significant [?].

Furthermore, the evolution of smart cities, crypto cities [?], Decentralized Autonomous Organizations (DAOs) [?], and other algorithmic governance models are intrinsically linked to the advancement of electronic voting systems. Despite the pressing need for such systems, as evidenced by countries such as Switzerland [?] and Estonia [?], global progress in adopting these modern democratic tools lags behind other digital transformations.

The feasibility of Internet voting has been the subject of extensive research, particularly in the field of cryptography, a critical aspect of system security. However, scepticism about the viability of public internet voting remains [?, ?, ?, ?, ?, ?]. In Germany, for example, the development of e-voting was halted following a court ruling against electronic voting machines, citing their contradiction with the public nature of elections [?]. The reluctance to use e-voting stems from issues of trust in the technology and the need for authoritative control over the voting process.

Criticism of Internet voting tends to concentrate on two arguments:

1. The inherent imperfection of the software, which precludes absolute trust.
2. Excessive reliance on centralised authorities to oversee the voting process.

Recent research from MIT suggests that any paperless voting system is inherently flawed [?]. Even high quality software, in the 90th percentile of the industry, contains an average of one defect per ten thousand lines of code [?]. These defects can lead to either malfunctions or exploitable vulnerabilities, potentially compromising the election process. The critical concern is that software defects should not result in undetectable changes to election results, a guarantee that seems unachievable given the nature of software development. However, recent developments in cryptography, in particular zero-knowledge proofs, offer promising solutions for ensuring the integrity of voting software [?]. Zero-knowledge proofs allow public verification of the correctness of the voting process without compromising voter privacy. This approach, which uses cryptographic verification, is consistent with the software independence requirement outlined in [?], allowing third-party verification without relying on the internal software of the voting system.

However, trust in the software is only part of the equation; the reliability of the hardware used by voters is also crucial. Critics argue that the security of Internet voting protocols depends on the assumption that voter devices are uncompromised and function as intended, a premise often considered unrealistic [?]. However, despite vulnerabilities in hardware, including trusted platforms that have been compromised [?,?,?], there is a trend towards improving hardware security, suggesting a positive trajectory in cybersecurity [?].

Furthermore, Appel et al. [?] highlight that no vote counting method is infallible, whether it is optical scanning, touch screen systems or manual counting. The critical issue is not the absence of security, but rather the degree of security and the nature of the trust assumptions involved.

The second major criticism of Internet voting relates to trust in the authorities overseeing the process. Traditional polling stations, being physically accessible and observable, provide a form of evidence-based trust that digital polling stations - servers - lack. The principle of evidence-based elections requires not only the determination of the winner, but also the provision of convincing evidence of that outcome to the electorate [?]. This evidence is considered convincing if the electoral process is both auditable and verifiable; it must produce a reliable audit trail and be routinely audited as part of the electoral process.

Ideally, the voting process should be completely trustless, meaning that there should be no trust assumptions other than our perceptions.

In reality, full monitoring of elections is impractical, leading to a reliance on designated staff to oversee the process. This approach is consistent with the 1 of N trust model, where the integrity of the system depends on the presence of at least one honest observer among N to report any discrepancies [?]. However, as the number of observers decreases, the likelihood of having at least one honest observer decreases, thereby compromising the trustworthiness of the election. Consequently, a robust voting process should involve a large, diverse group of observers - the larger the N, the more credible the process. Criticism of centralised Internet voting systems often focuses on their reliance on a strict 1 of 1 trust model. This model implies a single point of failure: the central authority. If this authority is compromised, then all trust in the system collapses, as it cannot provide voters with convincing evidence of the correctness of the software.

There are two counter-arguments to this criticism:

1. The first counter-argument is that even in the absence of trust in the organisers, requiring them to produce a cryptographic proof of the vote tally ensures accuracy. Any discrepancy in the results would be detectable in the verification of that proof.
2. The second counter-argument suggests decentralising the traditional centralised authority. This can be achieved through distributed systems that change the trust model from 1 of 1 to more robust models. These include N of N, where all participants must function correctly; Few of N, where a subset of nodes must be reliable; or $\frac{N}{2}$ of N, where the system remains functional as long as a majority of the nodes work correctly.

When trust assumptions in these systems are violated, various properties may fail, such as liveness, security, resistance to censorship, privacy, or correctness.

Blockchain technology has gained prominence in decentralising trust by providing inherent guarantees of immutability, verifiability, integrity, and resistance to censorship. As a result, blockchain has become a popular foundation for the development of trust-minimising platforms, including internet voting protocols.

Even in scenarios where blockchain is not explicitly used, and instead a distributed set of authorities (sometimes referred to as Guardians [?]) manage the voting system—as seen in Helios [?] or ElectionGuard [?]—these systems still require a mechanism to ensure that the Guardians themselves are trustworthy and that their collective actions result in a reliable and verifiable election outcome.

In this paper, we present PeerVote: a novel peer-to-peer voting protocol that employs a generalised trust model that allows participants to autonomously choose their trusted guardians within the system.

Figure 1 illustrates the evolution of trust models in Internet voting, from reliance on a single trusted third party to distributed third parties, and culminating in the peer-to-peer and delegated voter-to-voter models that we explore here.

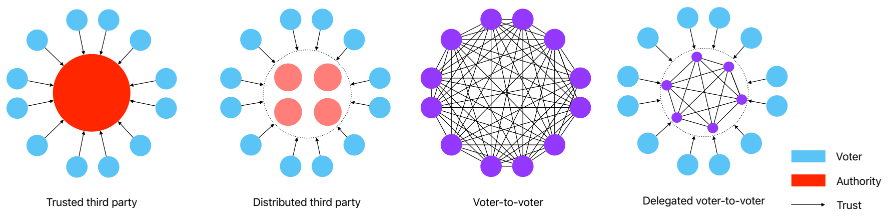

Fig. 1: Four trust models: trusted third party, distributed third part, voter-to-voter, and delegated voter-to-voter

Protocols using a distributed set of authorities typically use Distributed Key Generation (DKG) [?] to jointly generate encryption keys. The integrity of the voting process is maintained under the assumption of an 'honest majority', where privacy is preserved provided there is no collusion between the parties. However, the traditional DKG model faces challenges in voting systems, particularly when parties involved in the DKG are unavailable, potentially disrupting the voting process [?]. While this may not be an issue for government-level elections, in smaller elections collateral deposits may be required to incentivise participation [?], we find this approach impractical for wider applications. Instead, we propose a protocol based on threshold cryptography that tolerates a certain level of node unavailability. Our contributions are as follows:

1. We present an Internet voting protocol based on a delegated voter-to-voter trust model. This model reflects the trust relationships within a group and expresses them in the security and availability properties of the system. It is

particularly suitable for small-scale elections where participants are familiar with each other and existing trust lines are established.
2. We develop a new technique for dynamic Distributed Key Generation (DKG), called Federated DKG (FDKG). FDKG facilitates joint key pair generation by members with a single message exchange, without prior knowledge of all participants.
3. We propose a solution to the problem of node unavailability, which can hinder the election process. Using threshold cryptography and the concept of Guardian Sets, each participant shares its secret with a selected set of trusted nodes. Provided that these Guardian Sets consist of reputable and reliable network members, the risk of failure or collusion is minimised. This ensures that the improved availability of our protocol does not compromise its security.
4. We outline a practical implementation strategy for a peer-to-peer environment with no transaction fees, and an alternative public blockchain implementation using a single paymaster. This approach offers a significant advantage over existing protocols that require hosting or per-vote fees.
5. We present an open source implementation of our protocol in TypeScript and Golang, improving its cross-platform adaptability and thereby increasing its accessibility and practical utility.

2 Related Work

Internet voting protocols typically rely on a trusted third party, with variations in server capabilities determining integrity, anonymity, privacy, censorship resistance, and coercion resistance based on the trustworthiness of that entity. Current research mainly uses blockchain technology for its integrity and transparency in vote storage, as seen in systems such as Voatz, Polys, Follow My Vote, Verify-Your-Vote, OpenVoteNetwork, TIVI, Stellot, Cicada, Aragon/Vocdoni and MACI [?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?].

Alternatively, projects such as Helios, Civitas, Swisspost/Scytl, CHVote, iVoting and ElectionGuard use distributed authority and multi-party computation (MPC) protocols without explicitly relying on blockchain technology [?, ?, ?, ?, ?]. These systems are not fully decentralised, relying on a closed set of trusted entities known as Guardians.

Public blockchain-based solutions typically struggle with usability and require voters to pay transaction fees. Non-blockchain solutions operate either in a SaaS model or as self-hosting software, shifting the operational burden to election organisers and potentially creating accessibility barriers for small-scale voting and non-technical operators. Hybrid systems, like MACI, require both a blockchain network and a single coordinator server [?, ?].

Our proposed solution differs significantly from these models. It is free, unlike public blockchain-based systems, potentially increasing voter participation. Unlike private blockchain-based systems, our protocol operates without centralised servers, making it particularly suitable for smaller, informal voting settings. This approach shifts the trust paradigm from central authorities to the voters themselves.

Property	**Central server**	**Private network**	**Public blockchain**	**Voter-to-voter network**
Transaction fees	No	No	Yes	No
Service cost	Medium	High	No	No
Ease of use	High	High	Low	Medium
Trust to	Central authority	Authorities	Miners	Voters

Table 1: A comparative analysis of four types of Internet voting protocols: central server, private network, public blockchain, and voter-to-voter network, highlighting their differences in transaction fees, service costs, ease of use, and trust dynamics, where trust refers to an entity that guarantees properties of censorship resistance, privacy, and correctness. Examples of authorities are: "Returning Officers, members of the Board of Trustees, government officials or other trusted authorities who are responsible and accountable for the conduct of the election" [?]. The service cost includes all costs related to running the software like implementation, maintenance, and fees.

Furthermore, various systems achieve different security and privacy properties depending on their underlying assumptions. For example, OpenVoteNetwork eliminates the need for trusted third parties by adopting a self-tallying scheme, which ensures perfect ballot secrecy but is vulnerable to denial of service attacks [?, ?, ?, ?]. Cicida achieves ballot secrecy using homomorphic time-lock puzzles and voter anonimity using Semaphore for zero-knowledge set membership proofs [?, ?]. Aragon/Vocdoni, like Cicida, achieves voter anonymity using zero-knowledge set membership proofs, but election secrecy is achieved using trusted nodes called KeyKeepers. When a new election is created, each KeyKeeper creates an encryption key and publishes the public part via a transaction (setProcessKeysTx). The voters select up to N (at least one) of the keys and encrypt the vote with the N keys in onion mode [?, ?]. Minimal Anti-collusion Infrastructure (MACI), guarantees the highest level of security, i.e., censorship resistance, privacy, voter anonymity, and coercion resistance, however, the ballot privacy relies on a single coordinator server [?, ?].

ElectionGuard distributes encryption keys among Guardians, using Distributed Key Generation and multi-party computation for decryption, with a quorum parameter to ensure completion even if some Guardians are absent [?].

Our protocol follows ElectionGuard's approach, but extends it by allowing each voter to act as a Guardian, sharing their secret input with trusted parties instead of the entire network. This model not only further decentralises trust, but also enhances privacy and security by distributing the power of vote verification among a wider set of participants.

Objectives In designing the protocol, we focused on key objectives to address the essential challenges and needs of a secure and practical Internet voting protocol. These objectives are

1. **Distributed Model.** Use a fully distributed model to eliminate the need for central authorities or trusted intermediaries. This approach increases the resilience of the system to attack and reduces the risk of single points of failure.
2. **Flexible Trust.** Use a social mapping approach to security, where the system relies on the structure of trust within the community. People who are more trusted should have a greater role in maintaining security.
3. **Privacy.** Ensure voter anonymity and vote secrecy using cryptographic methods based on the honest-trust-majority assumption. This means that individual votes remain secret and cannot be decrypted unless the most important nodes collude.
4. **Robustness**: The protocol should be robust to partial participation, especially in the final round, allowing for flexible participation without compromising the integrity of the voting process.
5. **Accessibility**. Ensure that the protocol works smoothly on common devices such as smartphones and laptops. This makes it accessible, scalable and easy to use for everyone.

3 Preliminaries

In this section, we outline essential cryptographic concepts and tools that underlie the PeerVote protocol presented in this paper. These preliminaries provide the necessary background for understanding the structure and functionality of our protocol.

3.1 zk-SNARK

zk-SNARKs are cryptographic tools that allow one party (the prover) to prove to another (the verifier) the truth of a statement without revealing any information beyond the validity of the statement [**?**].

Consider an arithmetic circuit C characterised by a relation $\mathcal{R}_C$ and a language $\mathcal{L}_C$. This circuit accepts a statement $\vec{s}$ and a witness $\vec{w}$ such that $(\vec{s}, \vec{w}) \in \mathcal{R}_C$. A zk-SNARK for the satisfiability of this circuit is defined by three polynomial-time algorithms [**?**, **?**]:

- (pk,vk) $\leftarrow$ Setup($1^\lambda, C$). For a given security parameter λ and the circuit C, this algorithm produces a common reference string (CRS) comprising a proving key pk and a verifying key vk, both of which are public parameters associated with the circuit C.

Table 2: Summary of Notations

Notation	Description
$\mathbb{P}$	Set of all parties $P_i \in \mathbb{P}$ in the voting process
P_i,s_i	Public and secret keys used to authenticate i-th party, where $P_i = s_iG$
$\mathbb{D} \subseteq \mathbb{P}$	Subset of parties involved in the 1. FDKG phase
E_i,d_i	Partial encryption (public) and decryption (secret) keys of P_i. $E_i = d_iG$, where d_i is a random ephemeral (per vote) scalar value.
$\mathbf{E}$, $\mathbf{d}$	Voting public (encryption) and secret (decryption) keys, that is a sum of partial encryption keys E_i, and decryption keys d_i accordingly
$\mathbb{G}_i$	Guardian set is a subset of parties selected by P_i that can recreate P_i's part of the decryption key d_i. $\mathbb{G}_i \subseteq \mathbb{P} \setminus P_i$
$k = \lvert\mathbb{G}\rvert$	Total number of parties in a guardian set
t	Threshold number to reconstruct the shared secret
$[d_i]_j$	Share of partial decryption (secret) key, from P_i to P_j
$\mathbb{V} \subseteq \mathbb{P}$	Subset of parties participating in the 2. Voting phase
v_i	Encoded vote of P_i
r_i	One-time random value used for secure ElGamal encryption
$B_i = (C1_i, C2_i)$	Encrypted ballot of participant P_i using ElGamal scheme consisting of $(C1_i, C2_i)$, where $C1_i = r_iG$, and $C2_i = r_i\mathbf{E} + v_iG$
$\mathbb{T} \subseteq \mathbb{P}$	Subset of parties participating in the 3. Online Tally phase
$C1$	Sum of the first parts of all the casted ballots, i.e., $C1 = \sum_{P_i \in \mathbb{V}} C1_i$
$C2$	Sum of the second parts of all the casted ballots, i.e., $C2 = \sum_{P_i \in \mathbb{V}} C2_i$
PD_i	Partial decryption from P_i, i.e., $\text{PD}_i = d_iC1$
$[\text{PD}_i]_j$	Share of partial decryption, from P_i to P_j, i.e., $[\text{PD}_i]_j = [d_i]_jC1$
$\texttt{Enc}_{P_i}, \texttt{Dec}_{s_i}$	Public key encryption for P_i and decryption using the corresponding secret key s_i, as described in Appendix A
Δ	A helper value used to encode a scalar value to a point on a curve
$C_{i,j}$	Encrypted partial decryption key share, from P_i to P_j
PROOF_{B_i}	zkSNARK proof of correctness of B_i
$\text{PROOF}_{\text{FDKG}_i}$	zkSNARK proof of correctness of $(d_i,\ C_{i,})$
$\text{PROOF}_{\text{PD}_i}$	zkSNARK proof of correctness of PD_i
$\text{PROOF}_{[\text{PD}_i]_j}$	zkSNARK proof of correctness of $[\text{PD}_i]_j$

- $\pi \leftarrow \text{Prove}(\text{pk}, \vec{s}, \vec{w})$. Using the proving key pk, the statement $\vec{s}$, and the witness $\vec{w}$ such that $(\vec{s}, \vec{w}) \in \mathcal{R}_C$, this algorithm generates a non-interactive zero-knowledge proof π for the statement $\vec{s} \in \mathcal{L}_C$, demonstrating the relationship between $\vec{s}$ and $\vec{w}$.
- $0/1 \leftarrow \text{Verify}(\text{vk}, \vec{s}, \pi)$. With the verifying key vk, the statement $\vec{s}$, and the proof π, this algorithm outputs 1 if π is a valid proof for the statement $\vec{s} \in \mathcal{L}_C$, and outputs 0 otherwise.

3.2 Federated Distributed Key Generation

The Distributed Key Generation (DKG) protocol aims to collectively generate a voting encryption key pair without any single participant learning the secret (decryption) key. Each party $P_i \in \mathbb{P}$ learns only its share of this key, while the public (encryption) key is widely known. The protocol uses threshold cryptography, which allows flexibility in the participation of the participants during the tallying phase.

Secret Sharing Secret sharing via Shamir's Secret Sharing (SSS) scheme enables a dealer to distribute a secret key s over a randomly chosen polynomial $\mathbf{f}(X) = a_0 + a_1 X + a_2 X^2 + \cdots + a_{t-1} X^{t-1}$, where coefficients $a_0, a_1, \ldots, a_{t-1} \in_R \mathbb{F}_q$. Here, the secret key $s = a_0 = \mathbf{f}(0)$ and $t-1$ denotes the polynomial's degree. Shares are computed by evaluating $\mathbf{f}(i)$ for $i \neq 0$. Using Lagrange's Theorem, reconstructing $\mathbf{f}(X)$ and hence extracting $s = \mathbf{f}(0)$ is possible with t points on the polynomial.

Distributed Key Generation To avoid centralising the role of the dealer (and revealing the secret value s), the generation of polynomial $\mathbf{f}(X) \in_R \mathbb{Z}_q[X]$ is distributed among all parties $\mathbb{P}$. Each party selects a random polynomial $f_i(X) \in \mathbb{Z}_q[X]$, and the final polynomial is the sum of these individual polynomials:

$$\mathbf{f}(X) = \sum_{i=1}^{n} f_i(X)$$

Consequently, the voting secret (decryption) key $\mathbf{d}$ and voting public (encryption) key $\mathbf{E}$ are defined as:

$$\mathbf{d} = \mathbf{f}(0)$$

$$\mathbf{E} = \mathbf{d}G$$

To prevent arbitrary submissions by parties, Publicly Verifiable Secret Sharing (PVSS) is used, incorporating zero-knowledge proofs to validate the correctness of shared values [?].

Dynamic Distributed Key Generation Traditional DKG requires a known, fixed number of participants due to its reliance on SSS with predefined polynomial degrees. As we are aiming for an optional DKG phase with an unpredictable number of participants, we need a mechanism that allows dynamic adjustments to the number of participants.

Existing dynamic DKG schemes, such as the one in [?], necessitate continuous online presence of all parties, which we find impractical. Our objective is a non-interactive protocol where parties send only a single message and are then free to leave (You Only Speak Once approach [?]). We propose Federated DKG which facilitates joint key establishment by members with a single message, without prior knowledge of all participants. The technique works similar to the Federated Byzantine Agreement (FBA) used in the Stellar Consensus Protocol [?]. The details are described in the next section.

4 Voting Protocol

Our protocol integrates the three-round voting scheme from [**?**], the multi-candidate encoding method from [**?**], and the Federated Distributed Key Generation (FDKG) introduced in this paper.

Assumptions The protocol is based on the following assumptions:

1. All communications occur over a public message board available to all parties.
2. Parties are identified only by their public keys P_i and authenticated by their secret keys s_i.
3. Private channels over public message board are secured using encryption functions $\texttt{Enc}_{P_i}$ and $\texttt{Dec}_{s_i}$ described in Appendix A.
4. Each party verifies zkSNARK proofs and rejects messages that fail verification.
5. Participants consent to certain cryptographic parameters, including the BabyJubJub Elliptic Curve [**?**] $E(\mathbb{Z}_p)$ with a defined curve finite field modulus p, a base point G on the curve, and the order of the base point q. Additionally, they agree to the set of eligible voters $\mathbb{P}$ and the set of candidates.

4.1 Round 1: Federated Distributed Key Generation

Participation in the FDKG phase is optional. For each participating party $P_i \in \mathbb{D}$, where $\mathbb{D} \subseteq \mathbb{P}$:

1. A guardian set of k parties $\mathbb{G}_i \subseteq \mathbb{P}/P_i$ is selected based on the established trust lines of P_i.
2. A random polynomial $f_i(X) \in_\$ \mathbb{Z}_q[X]$ of degree $t-1$ is sampled.
3. Partial decryption (secret) key $d_i = f_i(0)$ and partial encryption (public) key $E_i = d_i G$ are computed.
4. A t-of-k access structure for d_i is created using PVSS. For each guardian $P_j \in \mathbb{G}_i$, partial decryption key share $[d_i]_j = f_i(j)$ is encrypted as $C_{i,j} = \texttt{Enc}_{P_j}([d_i]_j)$ as described in Appendix A.
5. Compute a zero-knowledge proof $\text{PROOF}_{\text{FDKG}_i}$, as described in Section 4.4.
6. Broadcast (E_i, $C_{i,j}$, $\text{PROOF}_{\text{FDKG}_i}$).

State after Round 1: Upon completion of FDKG, the message board contains:

- $\{E_i : P_i \in \mathbb{D}\}$, the set of partial encryption keys, where $\mathbf{E}$ can be reconstructed by anyone by summing $\mathbf{E} = \sum_{P_i \in \mathbb{D}} E_i$.
- $\bigcup_{P_i \in \mathbb{D}} \{C_{i,j} \mid P_j \in \mathbb{G}_i\}$, the set of encrypted shares of the partial decryption keys.

4.2 Round 2: Casting Votes

For each voter $P_i \in \mathbb{V}$, where $\mathbb{V} \subseteq \mathbb{P}$:

1. Encode a multi-candidate ballot using the method outlined in [**?**]. The encoding assigns a power of two to each candidate: a vote for candidate 1 as 2^0, for candidate 2 as 2^m, for candidate c as $2^{(c-1)m}$. The parameter m is selected as the smallest integer where $2^m > |\mathbb{P}|$. Thus, a vote is defined as

$$v_i = \begin{cases} 2^0 & \text{if } P_i \text{ votes for candidate 1} \\ 2^m & \text{if } P_i \text{ votes for candidate 2} \\ \vdots & \vdots \\ 2^{(c-1)m} & \text{if } P_i \text{ votes for candidate } c \end{cases}$$

2. Encrypt the vote using ElGamal encryption as $B_i = (r_i G,\ r_i\mathbf{E} + v_i G)$, where $r_i \in_{\$} \mathbb{Z}_q$ serves as a one-time blinding value.
3. Compute a zero-knowledge proof PROOF_{B_i}, as described in Section 4.4.
4. Broadcast $(B_i, \text{PROOF}_{B_i})$.

State after Voting Once the voting phase concludes (either when $|\mathbb{P}|$ messages have been received or after a predefined period), the message board's state is appended with:

- $\{B_i : P_i \in \mathbb{V}\}$, the set of encrypted votes.

4.3 Round 3: Tally

The tally process consists of two distinct phases: online and offline.

Online Tally The subset of parties $\mathbb{T} \subseteq \mathbb{P}$ involved in Threshold ElGamal decryption includes at least t parties from each guardian set $\mathbb{G}_1, \ldots, \mathbb{G}_{|\mathbb{D}|}$. For each party $P_i \in \mathbb{T}$:

1. Sum the first component of all ballots $C1 = \sum_{P_i \in \mathbb{V}} C1_i$, where $(C1_i, C2_i) = B_i$.
2. If $P_i \in \mathbb{D}$:
   (a) Compute the partial decryption $\text{PD}_i = d_i C1$.
   (b) Generate a zero-knowledge proof $\text{PROOF}_{\text{PD}_i}$ (as described in Section 4.4).
   (c) Broadcast $(\text{PD}_i, \text{PROOF}_{\text{PD}_i})$.
3. For each received encrypted share $C_{j,i}$, where $P_j \in \mathbb{D} \setminus \{P_i\}$ and $P_i \in \mathbb{G}_j$:
   (a) Decrypt to obtain $[d_j]_i = \mathtt{Dec}_{s_i}(C_{j,i})$.
   (b) Calculate the share of partial decryption $[\text{PD}_j]_i = [d_j]_i C1$.
   (c) Generate a zero-knowledge proof $\text{PROOF}_{[\text{PD}_j]_i}$ (outlined in Section 4.4).
   (d) Broadcast $([\text{PD}_j]_i, \text{PROOF}_{[\text{PD}_j]_i})$.

State after Online Tally After completing the Online Tally (the set is decryptable as defined in Definition 1), the message board's state is appended with:

– $\{\mathrm{PD}_i : P_i \in \mathbb{D} \wedge P_i \in \mathbb{T}\}$, the set of partial decryptions.
– $\bigcup_{P_i \in \mathbb{T}} \{[\mathrm{PD}_j]_i : P_j \in \mathbb{D} \setminus \{P_i\} \text{ and } P_i \in \mathbb{G}_j\}$, the set of shares of partial decryption.

Offline Tally The Offline Tally phase is accessible to anyone and involves the following steps:

1. Sum the second component of all ballots $C2 = \sum_{P_i \in \mathbb{V}} C2_i$, where $(C1_i, C2_i) = B_i$.
2. Calculate the sum of either partial decryptions or their reconstructions from shares:
$$Z = \sum \{\mathrm{PD}_i \text{ or } \sum([\mathrm{PD}_i]_j \lambda_j) : P_j \in \mathbb{G}_i\} = \mathbf{d}C1 = \mathbf{d} \sum_{P_i \in \mathbb{V}} r_i G$$
where $\lambda_i = \Pi_{j \neq i} \frac{j}{j-i}$ represents the Lagrange coefficient.
3. The decryption is $M = C2 - Z = (x_1 2^0 + x_2 2^j + \cdots + x_l 2^{(l-1)j})G$, where x_i is the number of votes for candidate i. It is because
$$\begin{aligned} M &= C2 - Z \\ &= (\sum_{P_i \in \mathbb{V}} r_i \mathbf{E} + \sum^{x_1} 2^0 G + \sum^{x_2} 2^j G + \cdots + \sum^{x_l} 2^{(l-1)j} G) - Z \\ &= \sum_{P_i \in \mathbb{V}} r_i \mathbf{E} + (x_1 2^0 + x_2 2^j + \cdots + x_l 2^{(l-1)j})G - \mathbf{d} \sum_{P_i \in \mathbb{V}} r_i G \\ &= \sum_{P_i \in \mathbb{V}} r_i \mathbf{E} + (x_1 2^0 + x_2 2^j + \cdots + x_l 2^{(l-1)j})G - \sum_{P_i \in \mathbb{V}} r_i \mathbf{E} \\ &= (x_1 2^0 + x_2 2^j + \cdots + x_l 2^{(l-1)j})G \end{aligned}$$
4. Extract each x_i by solving the Discrete Logarithm Problem. Given the small range of x_i ($0 \leq x_i \leq |\mathbb{V}|$), this is a feasible task. The extraction technique for each x_i follows the method described in [**?**].

4.4 Proofs Constructions

In our protocol, we employ the general principles of zk-SNARK to construct specific proofs for each phase of the voting process. Each proof is defined by a circuit C that verifies constraints, a public instance (statement) $\vec{s}$, and a private input (witness) $\vec{w}$.

PROOF$_{\mathbf{FDKG}_i}$

– **Circuit** (C): Defined as "Given E_i, $C_{i,j}$ and $P_j \in \mathbb{G}_i$, I know $f_i = a_0, \ldots, a_{t-1}$, $r1_1, \ldots, r1_k$, and $r2_1, \ldots, r2_k$, s.t. $E_i = a_0 G$ and the $C_{i,j}$ is an encrypted value of a polynomial f_i applied to j". The specifics of this circuit are detailed in Algorithm 1.

- **Public Instance** ($\vec{s}$): Includes the partial encryption key E_i, the guardian set $\mathbb{G}_i = \{P_1, \ldots, P_{|\mathbb{G}|}\}$, and the set of encrypted partial decryption key shares $\{C_{,1}, \ldots, C_{,|\mathbb{G}|}\}$.
- **Private Input** ($\vec{w}$): Includes the coefficients of the polynomial $\{a_0, \ldots, a_{t-1}\}$ and random values $\{r1_1, \ldots, r1_{|\mathbb{G}|}\}$, $\{r2_1, \ldots, r2_{|\mathbb{G}|}\}$.

PROOF$_{\boldsymbol{B_i}}$

- **Circuit** (C): Defined as "Given $\mathbf{E}$ and $B_i = (C1, C2)$, I know r_i, and v_i s.t. $v_i \in \{2^0, 2^j, \ldots, 2^{(l-1)j}\}$ and $B_i = (r_i G,\ r_i\mathbf{E} + v_i)$". Algorithm 2 elaborates on this circuit.
- **Public Instance** ($\vec{s}$): Includes the encryption key $\mathbf{E}$ and the encrypted ballot $B_i = (C1, C2)$.
- **Private Input** ($\vec{w}$): Includes the vote v_i and the random blinding factor r_i.

PROOF$_{\mathbf{PD}_i}$

- **Circuit** (C): Specified as "Given $C1, \mathrm{PD}_i, E_i$, I know a partial decryption key d_i s.t. $E_i = d_i G$ and $\mathrm{PD}_i = d_i C1$". Details of this circuit can be found in Algorithm 3.
- **Public Instance** ($\vec{s}$): Includes the sum of first ballot components $C1$, the partial decryption from party PD_i, and the partial encryption key E_i.
- **Private Input** ($\vec{w}$): Includes the partial decryption key d_i.

PROOF$_{[\mathbf{PD}_i]_j}$

- **Circuit** (C): Outlined as "Given $[\mathrm{PD}_j]_i, C_{j,i}, C1$, I know a secret key s_i s.t. $[\mathrm{PD}_j]_i = C1[d_j]_i$ where $[d_j]_i = \mathtt{Dec}_{s_i}(C_{j,i})$". For more information, refer to Algorithm 4.
- **Public Instance** ($\vec{s}$): Includes the sum of first ballot components $C1$, the share of partial decryption $[\mathrm{PD}_j]_i$, the encrypted partial decryption key share $C_{j,i}$, and the difference Δ.
- **Private Input** ($\vec{w}$): Includes the secret key of party s_i.

Algorithm 2: Circuit EncryptedBallot(m = the smallest integer s.t. $2^m > |\mathbb{P}|$)

```
Input: Statement $\vec{s} : (\mathbf{E}, B_i = (C1, C2))$
Output: Witness $\vec{w} : (v_i, r)$
1 assert $C1 = \mathtt{EscalarMulFix}(r, G)$;
2 F ← $\mathtt{EscalarMulAny}(r, \boldsymbol{E})$;
3 e ← $2^{(v_i - 1)m}$;
4 H ← $\mathtt{EscalarMulFix}(e, G)$;
5 assert $C2 = \mathtt{BabyAdd}(F, H)$;
```

Algorithm 1: Circuit FDKG(k = guardian set size, t = threshold)

```
Input: Statement $\vec{s} : (E_i, \mathbb{G} = \{P_1, \ldots, P_{|\mathbb{G}|}\}, \{C_{,1}, \ldots, C_{,|\mathbb{G}|}\})$
Input: Witness $\vec{w} : (\{a_0, \ldots, a_{t-1}\}, \{r1_1, \ldots, r1_{|\mathbb{G}|}\}, \{r2_1, \ldots, r2_{|\mathbb{G}|}\})$
assert $E_i$ = EscalarMulFix($s_i$, $G$);
for $i$ in $|\mathbb{G}|$ do
    eval[$i$][0] $\leftarrow a_0$;
    for $j$ in t do
        e $\leftarrow (i+1)^j$ % q;
        d $\leftarrow (a_j \cdot e)$ % q;
        eval[$i$][$j$] $\leftarrow$ (eval[$i$][$j-1$] + $d$) % q;
    $R \leftarrow$ EscalarMulAny($r1_i, P_i$);
    $F \leftarrow$ EscalarMulFix($r2_i, G$);
    $J \leftarrow$ BabyAdd($R, F$);
    $\Delta \leftarrow F_x -$ eval[$i$][t];
    assert $C_{,i}.C1$ = EscalarMulFix($r1[i]$, $G$);
    assert $C_{,i}.C2 = J$;
    assert $C_{,i}.\Delta = \Delta$;
```

Algorithm 3: Circuit PartialDecryption

```
Input: Statement $\vec{s} : (C1, \mathrm{PD}_i, E_i)$
Input: Witness $\vec{w} : (d_i)$
assert $E_i$ = EscalarMulFix($d_i$, $G$);
assert $\mathrm{PD}_i$ = EscalarMulAny($d_i$, $C1$);
```

Algorithm 4: Circuit PartialDecryptionShare

```
Input: Statement $\vec{s} : (C1, [\mathrm{PD}_j]_i, C_{j,i}, \Delta)$
Input: Witness $\vec{w} : (s_i)$
$X \leftarrow$ EscalarMulAny($s_i, C_{j,i}[1]$);
$(M_x, M_y) \leftarrow$ BabyAdd($C_{j,i}[2], -X$);
$[d_j]_i \leftarrow M_x - \Delta$;
assert $[\mathrm{PD}_j]_i$ = EscalarMulAny($[d_j]_i, C1$)
```

5 Security Analysis

This section analyses key properties of the PeerVote protocol, focusing on Decipherability, Privacy, Integrity and Availability as crucial aspects underpinning the security of the system. We discuss the conditions under which these properties are maintained or violated, particularly in the context of honest and dishonest parties within a distributed, multi-party computational framework.

Definition 1 (Decipherability). *Decipherability is achieved if for each participant $P_i \in \mathbb{D}$, either the participant publishes the full partial decryption, or at least t members of its guardian set $\mathbb{G}_i$ publishes their respective shares of partial decryption. Formally:*

$$\begin{aligned}\forall P_i \in \mathbb{D}, &\mathit{publish}(\mathrm{PD}_i) \lor \\ &\exists \mathbb{S}_i \subseteq \mathbb{G}_i : |\mathbb{S}_i| \geq t \land \forall P_j \in \mathbb{S}_i \mathit{publish}([\mathrm{PD}_i]_j)\end{aligned}$$

where $\mathit{publish}(\cdot)$ denotes the publication of either the full partial decryption PD_i or the share of partial decryption $[\mathrm{PD}_i]_j$ in the Online Tally phase.

Decipherability is guaranteed when a sufficient number of parties act honestly and fulfill their role in the decryption process. If the set of honest and available parties drops below the decipherability threshold, the protocol's ability to ensure decipherability is compromised, potentially leading to an inability to decrypt the final tally.

Definition 2 (Privacy). *Privacy is achieved if and only if there does not exist any collusion that achieves Decipherability. Formally, privacy is achieved when:*

$$\nexists \mathbb{A} \subseteq \mathbb{P} : (\mathit{Decipherability}(\mathbb{A}) \land \mathit{Colludes}(\mathbb{A}))$$

where $\mathit{Decipherability}(\mathbb{A})$ denotes that the set $\mathbb{A}$ achieves Decipherability, and $\mathit{Colludes}(\mathbb{A})$ indicates that the set $\mathbb{A}$ colludes.

Privacy depends on the assumption that there is no collusion that achieves Decipherability. The existence of such a subset could lead to a compromise of the confidentiality of the vote, as such parties could collude to decipher individual votes and count votes before the official tally.

Definition 3 (Censorship resistance). *Censorship resistance is achieved when the network accepts messages from all eligible voters.*

Censorship resistance should be guaranteed by the message board platform used to run the protocol. In practice, it's achieved by the consensus protocol and an honest majority of the parties running it. In Section 8 we discuss possible instantiations of the message board.

Definition 4 (Integrity). *Integrity ensures that votes are counted accurately and cannot be altered by unauthorised parties. Integrity is achieved as long as everyone verifies all messages from other parties and the eligibility of the voters submitting the ballot. In addition, this property relies on the security of all cryptographic primitives used in the protocol.*

Integrity is maintained through the verification of proofs of correctness. Robust cryptographic mechanisms, including secure pseudorandom number generators, digital signatures, ElGamal encryption and zkSNARKs, are used to ensure integrity, ensuring that any unauthorised changes are detectable.

6 FDKG Example

To illustrate the FDKG process and highlight its improvements over traditional methods, consider a scenario involving a set of parties $\{P_1, \ldots, P_{10}\}$. In this example, we have $k = 3$ and $t = 2$. A subset of these parties, namely $\mathbb{D} = \{P_1, P_3, P_5, P_7, P_9\}$, participates in the FDKG, each forming their respective guardian set. This setup is shown in Figure 2.

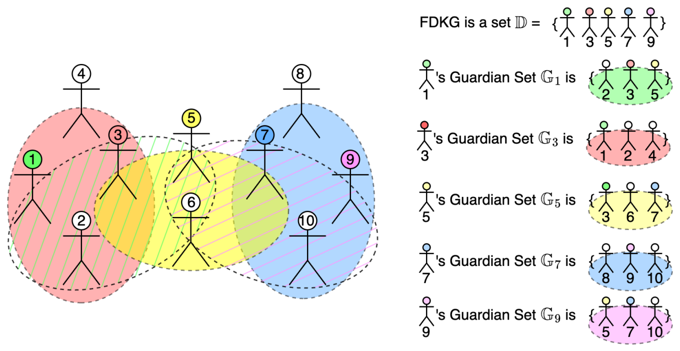
FDKG is a set $\mathbb{D}$ = {1 3 5 7 9}
1's Guardian Set $\mathbb{G}_1$ is {2 3 5}
3's Guardian Set $\mathbb{G}_3$ is {1 2 4}
5's Guardian Set $\mathbb{G}_5$ is {3 6 7}
7's Guardian Set $\mathbb{G}_7$ is {8 9 10}
9's Guardian Set $\mathbb{G}_9$ is {5 7 10}
1 2 3 4 5 6 7 8 9 10

Fig. 2: Example of Federated Distributed Key Generation

The FDKG process for each participant is as follows:

1. **Party** P_1:
   - Chooses Guardian Set $\mathbb{G}_1 = \{P_2, P_3, P_5\}$.
   - Samples a random polynomial $f_1(X) \in_R \mathbb{Z}_q[X]$, computes decryption key $d_1 = f_1(0)$.
   - Distributes d_1 among $\mathbb{G}_1$, yielding shares $[d_1]_2, [d_1]_3, [d_1]_5$.
2. **Party** P_3:
   - Chooses $\mathbb{G}_3 = \{P_1, P_2, P_4\}$.
   - Samples $f_3(X)$, computes d_3, and distributes it among $\mathbb{G}_3$, yielding shares $[d_3]_1, [d_3]_2, [d_3]_4$.
3. **Party** P_5:
   - Chooses $\mathbb{G}_5 = \{P_3, P_6, P_7\}$.
   - Samples $f_5(X)$, computes d_5, and distributes it among $\mathbb{G}_5$, yielding shares $[d_5]_3, [d_5]_6, [d_5]_7$.
4. **Party** P_7:
   - Chooses $\mathbb{G}_7 = \{P_8, P_9, P_{10}\}$.
   - Samples $f_7(X)$, computes d_7, and distributes it among $\mathbb{G}_7$, yielding shares $[d_7]_8, [d_7]_9, [d_7]_{10}$.
5. **Party** P_9:
   - Chooses $\mathbb{G}_9 = \{P_5, P_7, P_{10}\}$.
   - Samples $f_9(X)$, computes d_9, and distributes it among $\mathbb{G}_9$, yielding shares $[d_9]_5, [d_9]_7, [d_9]_{10}$.

In a traditional DKG protocol, the minimum set of parties required to achieve Decipherability would be $\{P_1, P_3, P_5, P_7, P_9\}$. However, with the FDKG approach and the use of Guardian Sets, this minimum set is reduced to $M = \{P_3, P_5, P_6\}$, demonstrating the protocol's efficiency in reducing the number of participants required for successful decryption.

Decipherability The set M ensures Decipherability because the shares d_3, d_5, d_6 are published directly by these parties, while d_1 is recoverable by shares $\{P_3, P_5\}$ and d_9 is recoverable by shares $\{P_5, P_7\}$.

Privacy Privacy in the FDKG system is compromised when all parties from any Decipherability set collude. In this example, if all parties $\{P_3, P_5, P_6\}$ collude, they can collectively reconstruct the secret $\mathbf{d}$ and therefore decrypt each individual ballot B_i.

7 Experimental Results

This section evaluates the PeerVote protocol, focusing on proof times and message sizes for two prominent proof systems: Plonk and Groth16, as implemented in the `snarkjs` WebAssembly (WASM) framework. The execution time for solving the Discrete Logarithm Problem (DLP) in the Offline Tally phase was also evaluated. These experiments were performed on a MacBook Pro with an Apple M1 Pro processor and 16GB of RAM.

Proving Time Table 3 shows the time taken to generate a zkSNARK proof for each message using Groth16 and Plonk. The results show a significant performance difference between the two, with Groth16 showing superior efficiency. In particular, during the FDKG phase, Groth16 required a maximum of 2.414 seconds for a 3 out of 4 configuration, compared to Plonk's 146.026 seconds for the same setup.

Proving System	**FDKG**			**Encrypt Ballot**	**Partial Decryption**	**Partial Decryption Share**
	1 of 2	**2 of 3**	**3 of 4**			
Groth16	1.388 s	2.135 s	2.414 s	0.747 s	0.619 s	0.580 s/share
Plonk	67.753 s	71.902 s	146.026 s	16.822 s	16.543 s	8.137 s/share

Table 3: Proving Time

Message Size Table 4 details the message sizes required in each round of the protocol. Messages using Groth16 are smaller due to their proofs comprising only 3 elliptic-curve points, whereas Plonk-based proofs require 9 points and 6 scalars [**?**].

Proving System	**FDKG**			**Encrypt Ballot**	**Partial Decryption**	**Partial Decryption Share**
	1 of 2	**2 of 3**	**3 of 4**			
Groth16	1152 B	1568 B	1984 B	512 B	576 B	768 B/share
Plonk	1472 B	1888 B	2304 B	832 B	896 B	1088 B/share

Table 4: Sizes of the Messages in Each Round

Discrete Logarithm Problem (DLP) A critical part of the Offline Tally in our protocol involves solving the DLP to extract the number of votes for each candidate. Although typically infeasible, the DLP can be solved by exhaustive search for a small number of voters. Using the method described in [**?**], we extracted each x_i from the final output point M.

Figure 3 shows that solving the DLP follows a power law relationship $Y = aX^b$, where b varies linearly with the number of options. This finding suggests that while the time to solve the DLP is linear in the number of voters, it grows exponentially with the number of candidates, posing scalability challenges in larger elections.

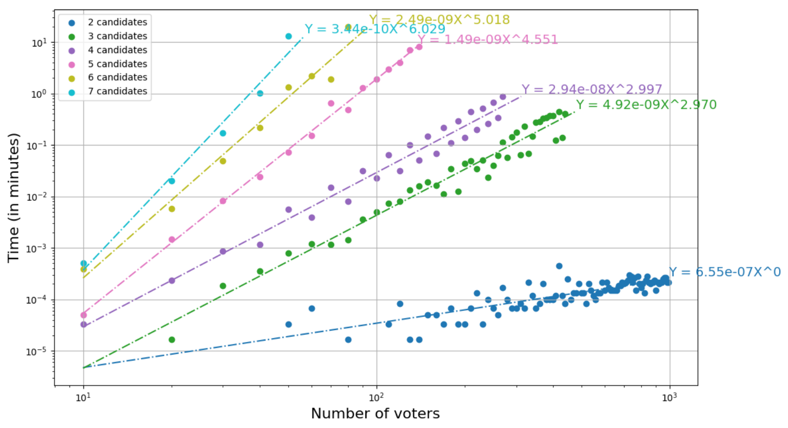

Fig. 3: Time required to solve DLP with respect to the number of candidates and number of voters (log-log scale).

8 Deployments

The protocol can be deployed on any platform that supports message board functionality, i.e. it provides a shared communication space where everyone can post messages and read them all in the same order. The ideal concept of a message board is difficult to achieve in practice due to the unreliable nature of distributed systems, especially in the presence of Byzantine nodes. However, there are many approximations. In particular, blockchain is the most promising, as it provides the highest level of security in a trustless environment. Blockchain comes at the cost of high transaction fees that all interacting parties have to bear. This is the price we want to avoid in democratic voting. In the future, as the standardisation of the ERC-4337 [?] progresses, transaction fees could be centralised under a single paymaster (e.g. the organiser), allowing gasless voting for participants.

A viable option are peer-to-peer networks. We find a promising framework for our protocol in the Wesh Network[1], a toolkit based on the libp2p stack for building peer-to-peer applications. It is designed for use in a mobile environment, focusing on ease of use, resilience and integrity.

Finally, the PeerVote protocol can be deployed on an centralised messenger platform such as Telegram, Signal or WhatsApp, where the message board is the chat room. This approach is less secure than the previous two, as the central server has potential control over the censorship and order of messages.

It's worth noting that PeerVote deployed on a central instant messaging platform still provides a high level of security and privacy as all messages are encrypted and authenticated, only the censorship property can potentially be exploited by the central server.

See Figure 4 for the conceptual stack of the protocol.

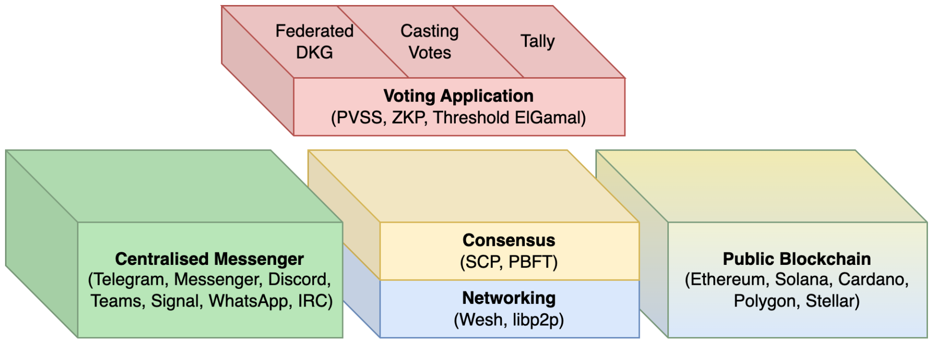
Federated DKG
Casting Votes
Tally
Voting Application
(PVSS, ZKP, Threshold ElGamal)
Centralised Messenger
(Telegram, Messenger, Discord, Teams, Signal, WhatsApp, IRC)
Consensus
(SCP, PBFT)
Networking
(Wesh, libp2p)
Public Blockchain
(Ethereum, Solana, Cardano, Polygon, Stellar)

Fig. 4: Three possible deployments of the protocol: Centralised messenger, ad-hoc peer-to-peer network, and public blockchain.

[1] Wesh Network, asynchronous mesh network protocol powered by Berty Technologies' non-profit organisation, `https://wesh.network/`

9 Discussion and conclusion

Our work introduces a robust voting protocol that eliminates reliance on centralised entities and demonstrates resilience to node unavailability. Federated Distributed Key Generation (FDKG) is our main innovation, extending traditional Distributed Key Generation (DKG) methods. FDKG uniquely allows key reconstruction by parties within their Guardian sets, in addition to those participating in the original DKG. This generalisation of DKG—where standard DKG is equivalent to a 0-of-k FDKG protocol—significantly improves the robustness of the protocol by minimising the size of the decipherability sets while preserving privacy, where these sets include the most trusted parties in the network.

The protocol's robustness is particularly valuable in contexts with uncertain network reliability and participant availability, ensuring secure and uninterrupted voting processes. Future research could include simulating real-world trust dynamics in communities to assess the applicability of the protocol [?].

However, implementing zkSNARKs for message attestation involves a trade-off between setup complexity and computational efficiency. Trusted setup ceremonies, such as those used in Groth16 [?], offer greater efficiency compared to transparent setups such as Plonk [?]. Our experiments showed that on a high-end personal computer, Groth16's verification time per message was 1-3 seconds, while Plonk required 16-145 seconds. Although trusted setups involve strong assumptions, they are a one-time requirement and can be reused in future votes, a common practice in zkSNARK-based protocols.

A critical aspect of our protocol is offline tallying, which relies on solving the Discrete Logarithm Problem (DLP). The complexity of this task follows a power law $Y = aX^b$, where X is proportional to the number of voters and b increases with the number of candidates, implying linear growth with the number of voters and exponential growth with the number of candidates. The computational challenge can be partially mitigated by parallelizing the exhaustive search over all parties. The overall time complexity would be reduced to $\frac{1}{n}$ time. Overcoming this scalability challenge remains an area for future improvement, possibly through the development of more efficient algorithms.

Three deployment options for our protocol are proposed (see Figure 4): as a peer-to-peer application using Wesh Network, or as a smart contract on public blockchains using ERC-4337 [?] to centralise voting costs under a single paymaster (e.g., organiser), allowing gasless voting for participants, or via a centralised messenger application such as Telegram, WhatsApp, Signal or Discord.

We have open sourced the implementation of our protocol in TypeScript and Golang to facilitate widespread use and improve accessibility. The codebase is available at `https://github.com/anonymised/for/review`

Optimization avenues include improving proof times and sizes by batching proofs or reformulating them into more efficient Sigma Proofs. Another path involves on-chain implementation with gasless transactions, leveraging ERC-4337's paymaster concept.

In the future, the anonymity of the voters could be additionally protected from collusion by the malicious parties by using zero-knowledge set membership proofs [?], as seen in protocols like Cicada [?] and Vocdoni [?]. In this way, although the colluding parties could decrypt individual ballots, they could not determine who had cast them.

In conclusion, our protocol represents a significant step forward in the evolution of Internet voting systems. By embracing decentralisation, enhancing security and prioritising privacy, we aim to contribute to the development of more resilient, trustworthy and inclusive voting mechanisms in the digital age. As we continue to refine and optimise our protocol, we remain focused on the broader goal of modernising democratic processes and empowering communities with reliable and accessible voting technologies.

Acknowledgements

We would like to thank Lev Soukhanov for his crucial role in designing the voting protocol and the Delendum team for their support and valuable insights during our research fellowship. We are grateful to Maya Dotan for her detailed review and helpful feedback, which significantly improved the quality and clarity of our work.

References

1. Building Cicada: Private on-chain voting using time-lock puzzles. https://a16zcrypto.com/posts/article/building-cicada-private-on-chain-voting-using-time-lock-puzzles/
2. CHVote 2.0 project release. https://chvote2.gitlab.io/
3. ERC-4337: Account Abstraction Using Alt Mempool. https://eips.ethereum.org/EIPS/eip-4337
4. Polys online voting system - Whitepaper. http://polys.vote/whitepaper
5. Semaphore. https://semaphore.pse.dev/
6. TIVI powered by Smartmatic and Cybernetica. https://tivi.io/
7. Vocdoni introduction | Vocdoni developer portal. https://developer.vocdoni.io/protocol/overview
8. Now You Can Vote Online with a Selfie. https://www.businesswire.com/news/home/20161017005354/en/Now-You-Can-Vote-Online-with-a-Selfie (Oct 2016)
9. Intel SGX Broken by "Plundervolt" Attack (Dec 2019)
10. ElGamal encryption, decryption, and rerandomization, with circom support - zk-s[nt]arks. https://ethresear.ch/t/elgamal-encryption-decryption-and-rerandomization-with-circom-support/8074 (Oct 2020)
11. A16z/cicada. a16z (Dec 2023)
12. Electronic voting by country. Wikipedia (Oct 2023)
13. Electronic voting in Estonia. Wikipedia (Sep 2023)
14. Electronic voting in Switzerland. Wikipedia (Nov 2023)
15. Privacy-scaling-explorations/maci. Privacy & Scaling Explorations (Dec 2023)
16. Secure Decentralized Application Development. https://followmyvote.com/ (May 2023)

17. Adida, B.: Helios: Web-based Open-Audit Voting. In: USENIX Security Symposium. vol. 17, pp. 335–348 (2008)
18. Appel, A.W., Stark, P.B.: Evidence-Based Elections: Create a Meaningful Paper Trial, Then Audit. Geo. L. Tech. Rev. **4**, 523 (2019)
19. Barański, S., Szymański, J., Sobecki, A., Gil, D., Mora, H.: Practical I-Voting on Stellar Blockchain. Applied Sciences **10**(21), 7606 (Oct 2020). https://doi.org/10.3390/app10217606
20. Baudron, O., Fouque, P.A., Pointcheval, D., Stern, J., Poupard, G.: Practical multi-candidate election system. In: Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing. pp. 274–283 (2001)
21. Benaloh, J., Naehrig, M.: ElectionGuard - Design Specification 2.0 (Aug 2023)
22. Bulck, J.V., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution p. 19
23. Buterin, V.: Trust Models (2020)
24. Buterin, V.: Blockchain voting is overrated among uninformed people but underrated among informed people (2021)
25. Buterin, V.: Crypto Cities (2021)
26. Chaieb, M., Yousfi, S., Lafourcade, P., Robbana, R.: Verify-Your-Vote: A Verifiable Blockchain-Based Online Voting Protocol. In: Themistocleous, M., Rupino Da Cunha, P. (eds.) Information Systems, vol. 341, pp. 16–30. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-11395-7_2
27. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: 2008 IEEE Symposium on Security and Privacy (Sp 2008). pp. 354–368. IEEE (2008)
28. Delerablée, C., Pointcheval, D.: Dynamic threshold public-key encryption. In: Advances in Cryptology–CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings 28. pp. 317–334. Springer (2008)
29. ElSheikh, M., Youssef, A.M.: Dispute-free Scalable Open Vote Network using zk-SNARKs (2022)
30. Ethereum Foundation: Minimal Anti-Collusion Infrastructure. Privacy & Scaling Explorations (formerly known as appliedzkp) (Aug 2022)
31. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge (2019)
32. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive (2019)
33. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In: Goos, G., Hartmanis, J., Van Leeuwen, J., Stern, J. (eds.) Advances in Cryptology — EUROCRYPT '99, vol. 1592, pp. 295–310. Springer Berlin Heidelberg, Berlin, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-$X_2$1
34. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: YOSO: You Only Speak Once. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021. pp. 64–93. Lecture Notes in Computer Science, Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_3
35. Golomb, G.: Believe It: Cybersecurity is Getting Better, Not Worse. https://www.infosecurity-magazine.com/opinions/cybersecurity-getting-better-worse/ (Jan 2018)

36. Goodin, D.: Intel SGX is vulnerable to an unfixable flaw that can steal crypto keys and more. https://arstechnica.com/information-technology/2020/03/hackers-can-steal-secret-data-stored-in-intels-sgx-secure-enclave/ (Mar 2020)
37. Goos, G., Hartmanis, J., Van Leeuwen, J., Schoenmakers, B.: A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99, vol. 1666, pp. 148–164. Springer Berlin Heidelberg, Berlin, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-$1_1$0
38. Groth, J.: On the Size of Pairing-based Non-interactive Arguments. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 305–326. Springer (2016)
39. Hao, F., Ryan, P.Y., Zieliński, P.: Anonymous voting by two-round public discussion. IET Information Security **4**(2), 62–67 (2010)
40. Heald, G.: A mathematical description of trust. Research Gate, available at: www. researchgate. net/publication/343119798_A_Mathematical_Description_of_Trust (2019)
41. Jie, K.W.: Weijiekoh/elgamal-babyjub (Dec 2023)
42. Laslier, J.F.: And the loser is... Plurality Voting (Jul 2011)
43. Lee, T.B.: Blockchain-based elections would be a disaster for democracy. https://arstechnica.com/tech-policy/2018/11/blockchain-based-elections-would-be-a-disaster-for-democracy/ (Nov 2018)
44. Llaguno, M.: 2017 Coverity Scan Report: Open Source Software—The Road Ahead. Tech. rep. (2017)
45. Mazieres, D.: The stellar consensus protocol: A federated model for internet-level consensus. Stellar Development Foundation **32**, 1–45 (2015)
46. McCorry, P., Shahandashti, S.F., Hao, F.: A Smart Contract for Boardroom Voting with Maximum Voter Privacy. In: Kiayias, A. (ed.) Financial Cryptography and Data Security, vol. 10322, pp. 357–375. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-70972-$7_2$0
47. Mearian, L.: Why blockchain-based voting could threaten democracy. https://www.computerworld.com/article/3430697/why-blockchain-could-be-a-threat-to-democracy.html (Aug 2019)
48. Moore, L., Sawhney, N.: The West Virginia Mobile Voting Pilot (2019)
49. Park, S., Specter, M., Narula, N., Rivest, R.L.: Going from bad to worse: From internet voting to blockchain voting. Journal of Cybersecurity **7**(1), tyaa025 (2021)
50. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE (2013)
51. Roenne, P.: JCJ with improved verifiability guarantees (2016)
52. Schneier, B.: Blockchain and Trust (2019)
53. Schneier, B.: On Blockchain Voting (2020)
54. Schoenmakers, B.: Lecture notes cryptographic protocols. Department of Mathematics and Computer Science, Technical University of Eindhoven, version **1** (2018)
55. Seifelnasr, M., Galal, H.S., Youssef, A.M.: Scalable Open-Vote Network on Ethereum (2020)
56. Shankland, S.: No, blockchain isn't the answer to our voting system woes. https://www.cnet.com/news/privacy/blockchain-isnt-answer-to-voting-system-woes/ (2018)
57. Wang, S., Ding, W., Li, J., Yuan, Y., Ouyang, L., Wang, F.Y.: Decentralized Autonomous Organizations: Concept, Model, and Applications. IEEE

Transactions on Computational Social Systems **6**(5), 870–878 (Oct 2019). https://doi.org/10.1109/TCSS.2019.2938190
58. WhiteHat, B., Baylina, J., Bellés, M.: Baby Jubjub elliptic curve (2020)
59. Williams, N.: Remote Voting in the Age of Cryptography. MIT Computational Law Report (Dec 2022)

A Private Channel

Each user uses a private, authenticated channel to send encrypted messages to other parties. We use ElGamal encryption on the BabyJub curve as described in [**?**, **?**]. The encryption process starts with a plaintext scalar m, which is transformed into a point on the BabyJub elliptic curve by generating a random value r, computing the message $M = rG$ and the x-increment Δ, which must be added to the plaintext to obtain the x-value of M. Thus, the ciphertext consists of two elliptic curve points and a field element $(C_1, C_2, \Delta = M.x - m)$. The complete encryption process is described in detail in Algorithm 5.

Decryption of the ciphertext (C_1, C_2, Δ) follows the standard ElGamal method, with the addition of Δ to the x-coordinate of the starting point, as described in Algorithm 6.

Algorithm 5: $\texttt{Enc}_{P_i}$

```
  Input  : A scalar $m$
  Output: A tripe $(C_1, C_2, \Delta)$
1 $k \leftarrow_R \mathbb{Z}_q$;
2 $r \leftarrow_R \mathbb{Z}_q$;
3 $C_1 = kG$;
4 $M = rG$;
5 $C_2 = kP + M$;
6 $\Delta = M.x - m$;
7 return $(C_1, C_2, \Delta)$
```

Algorithm 6: $\texttt{Dec}_{s_i}$

```
  Input  : A tripe $(C_1, C_2, \Delta)$
  Output: A scalar $m$
1 $M = C_2 - s_i C_1$;
2 $m = M.x - \Delta$;
3 return $m$
```