School of Electronic Engineering
and Computer Science

Final Report

Programme of study:
BSc Digital and Technology Solutions (Software Engineer)

Project Title:

Pnyx; A foundation for decentralised democracy without trusted parties

Supervisor:
Dr. Michael Shekelyan

Student Name:
George Yarnley

Final Year
Undergraduate Project 2023/24

Date: 2024-04-25

With thanks to Michael Shekelyan, Nathan Jeffrey and Emmanuel Lesi for their feedback and guidance, as well as all of my colleagues for their support

Abstract

Open source software has become fundamental to the entire software industry, underpinning almost every major project from hobbyists to industry leaders alike. However, open source communities regularly leverage non-representative and unstable leadership paradigms. This can often lead to instability including project abandonment, legal issues and vulnerability to bad actors.

This paper presents Pnyx; a system architecture and set of protocols representing a new approach for the governance of systematically important open source software projects. By combining the concepts of liquid democracy, version controlled documents and blockchain with the state of the art in zero-trust, decentralised computation we establish a foundation for improved open source software quality, reliability and security. The suggested system is composed of mature protocols and concepts, pulling together concepts from across the literature, and builds a proposal constrained by a number of desirable features established across similar projects. Additionally, some aspects of the proposal are explored further through a proof-of-concept implementation and a set of associated test scenarios. Finally, a number of important problems and subject areas requiring investigation to fully realise the proposed system are outlined to inform continued research and development.

Contents

Chapter 1: Background

Within the open source community, governance and objective alignment is a key issue. Projects are managed in many ways, from lone maintainers to dedicated foundations and even projects managed by for-profit entities.

In many instances these approaches have been sufficient. However, major projects have suffered unpopular changes to project aims, issues with inclusivity or abandonment of promising initiatives. For example, as described by Anderson (2023) in 'devclass', the Rust Foundation has been under fire twice: firstly due to the (later reversed) decision to trademark the Rust logo, and later on due to the decision to downgrade the keynote talk of a respected industry leader at the premier Rust conference 'RustConf'. These issues drove a number of key contributors away from the project (Turner, S; 2023), impacting long term development goals and reducing the speed at which issues could be addressed. Key figures attributed these failures to the decision making processes of 'Leadership Chat', an opaque governance structure controlled by a small group of individuals.

Another notable issue was HashiCorp's decision to re-license the Terraform project from the MPL to the BUSL, effectively removing it's open source status (OpenTofu, 2023). This decision created legal and operational risk for a huge number of products and tools leveraging Terraform as a part of their platform. Similar relicensing decisions have been made for many projects such as Redis and MongoDB (Colannino, 2021).

Additionally, few open source projects have meaningful plans in place for handling the loss of key contributors. This is extremely important in the case of projects with a very small maintainer pool. The xz utils backdoor in 2024 exposed the vulnerability of foundationally important projects maintained by small groups or even solo developers. Mensching (2024) emphasises how this is a systematic problem within the open source community that stems from a lack of community support and investment in critical projects.

All of these factors can represent a material risk for the entire industry. Individual developers and large international institutions alike rely heavily on this complex supply chain of software built on top of and around open source products. These issues are especially pertinent to key foundational projects which are depended on either directly or transiently by potentially thousands of downstream applications.

Democracy has demonstrated itself as one of the most powerful tools within modern society for coordinating large groups of individuals towards complex end goals. With the advent of open source as not only a technical and legal concept but a cultural phenomenon, the role of democratic governance has become ever more important. Unfortunately many open source projects either fail to effectively adopt democratic principles.

Partially, this is due to the only effective precedent available for democracy-at-scale being supremely cost prohibitive. For example, the 2015 UK General Election cost £114,732,548, an average cost of £1.57 per registered elector (HM Government. 2018). Even for a moderately sized open source project, this represents an insurmountable cost. On top of the issue of costs, issues can arise in terms of project and governance control, trust, and accumulated power.

1.1 Problem Statement

Within the open source community there is currently no solution enabling projects to set up highly trustworthy, ownerless, provably accurate democratic systems for project and community management. Existing tools either require trusting a privileged group, do not

provide substantive diffusion of power or are vulnerable to participation issues. The absence of such a solution invariably leads to projects using alternative governance structures, potentially reducing the long-term quality and reliability of important open source projects.

1.2 Aim

This project aims to lay the foundations for a system which would enable more stable and inclusive long term governance of systematically important open source projects. The solution must include a robust mechanism for collective decision making representative of the community's views. Such a system must also be decentralised, ownerless and trustless to mitigate the risk of misuse and cheating.

The initial specification for the system comprises a decentralised network which is able to immutably store votes against a given issue, and resolve those votes into results. These results are then persisted to the network in the form of a set of changes to a document which may represent regulations, guidelines, or intended development aims.

1.3 Objectives

1. Analyse existing electronic voting and democracy solutions, including existing approaches to decentralised voting
2. Discuss relevant technologies within the space of trustless systems such as Zero-Knowledge Proof, Blockchain, Consensus Algorithms, Encryption Schemes & Multi-Party Computation
3. Analyse and synthesise literature to suggest a model by which open source projects and other collectives may align goals, rules and regulations in a fair, democratic fashion
4. Develop and test software as a proof-of-concept to demonstrate the feasibility of the suggested solution
5. Analyse limitations of the proof-of-concept solution and proposed model
6. Recommend further work will be required to build a practical system, and identify any areas in which appropriate concepts are not yet available

Chapter 2: Context

2.1 Existing Electronic Voting Systems

A number of projects have approached the problem of an electronic approach to voting and democracy. One such example is the government of Estonia who have used internet-based voting systems since 2005 (Vassil et al., 2016). They have implemented internet voting protocols allowing individuals to cast their vote remotely and to be authenticated using public key cryptography. However, their approach has attracted criticism and scrutiny from a number of sources. Springall et al. (2014) published an analysis asserting that, if they were able to place malware on the vote counting server, they would be able to plausibly change votes, compromise voter secrecy, disrupt elections or otherwise damage the integrity of election results. The analysis further posited that these issues stemmed from 'basic architectural choices' and therefore were 'difficult to mitigate'. These analyses highlighted the potential vulnerabilities posed by trusting a central tallying authority.

In recent years, largely due to the surge in popularity of cryptocurrencies, an alternative set of e-voting schemes have emerged to serve Digital Anonymous Organisations (DAOs). Each of these systems relies on an underlying blockchain to provide confidence in the validity of votes, and to store vote results immutably in a decentralised manner. They also may leverage 'smart contracts' to execute tallying programs in a trust-free way. One example of this is 'Snapshot' (Snapshot Labs, 2023), which allows loose collectives of individuals to set up different voting contexts. Often this is used in a way mirroring traditional systems, electing leadership and community representatives. However, such systems are also able to provide community feedback and sentiment for 'RFC' style proposals.

There also exist proposed systems in the literature, such as ElectAnon (Onur & Yurdakul, 2022). Each of these systems relies on an underlying blockchain to provide confidence in the validity of votes, and to store vote results immutably in a decentralised manner. Additionally, each of them relies on a trusted authority which establishes the context for votes. This includes defining the eligible population, setting the topic of debate and establishing measures of power.

2.2 Democratic Systems

Within the literature democratic approaches traditionally fall in to one of two groups; representative democracy or direct democracy. In the former, members of a community select an individual or group to make decisions on behalf of the population. Generally, it is intended that these individuals act altruistically, or in the best interests of the overall population. Unfortunately, this is not always the case, as elective systems can suffer from corruption, lobbying and self-interest. Direct democracy, on the other hand, requires the whole population to voice their opinion on each issue as it occurs. However, ensuring the continuous engagement of the entire community is difficult to guarantee.

2.3 Liquid Democracy

An alternative approach to democracy is found in the literature in the form of Liquid Democracy (LD). As established in a review of the literature by Paulin (2020), LD is 'a way of making collective decisions [without] electing representatives'. Blum & Zuber (2016) model LD as a system with four properties:

1. 'Direct Democracy' - The ability for members of a community to vote directly on issues

2. 'Flexible Delegation' - The ability for members to delegate their vote to be cast by another in lieu of their own direct decision

3. 'Meta-Delegation' - The ability for delegated votes to be 're-delegated' by the selected delegate, forming a chain of delegations

4. 'Instant Recall' - The ability for members to recall or alter their delegation at any point in time, as well as to override their delegated verdict with a direct verdict

Liquid Democracy enables all members of the community to continuously participate in decisions which matter to them through the direct democracy component. This also works to reduce the concentration of power, leading to a more representative window into community intentions. On top of this, LD mitigates the risks of low turnout through the delegation components.

The primary difficulty in implementing a LD system is found in the complexity of resolving all of the delegations. Traditional voting systems are ill equipped to handle frequent, unpredictable changes in delegation or efficiently resolve the final vote weights whilst preserving privacy. For this reason, an electronic approach is necessary.

2.4 Existing Liquid Democratic Systems

LD has emerged as an attractive method of election within DAOs and other Web3 communities. The aforementioned 'Snapshot' allows for all the relevant qualities of a LD system, but is not a dedicated Liquid Democracy platform. Unfortunately, the 'Snapshot' system primarily supports one time isolated votes, and so leveraging the delegation capabilities can be cumbersome.

Paulin's review outlined a number of other systems implementing LD in more traditional political contexts, most notably LiquidFeedback. LiquidFeedback is the only system which has seen real use over an extended time frame. It received initial success within the voter base of the German Pirate Party. However, the system only provided a form of feedback to the elected officials and was non-binding. Officials 'actively [refusing]' to follow the directions given by the system led to 'an erosion of [its] relevance'.

Similar results can be observed in the literature for other systems, where a lack of trust in or engagement with a given system leads to a decline in its usage and its eventual obsolescence. This often seems to be due to the lack of real potency behind decisions made using the systems.

Chapter 3: Technological Basis

To achieve all of the constraints of the desired system, this project combines a wide variety of previously established technologies. The following establishes a baseline context for their utility and place within the literature.

3.1 Blockchain

A blockchain provides a way of storing data which is decentralised, immutable, censorship-resistant and guarantees eventual consistency. This makes it a great technology when building a source of truth which should not be controlled by any one party.

The term 'blockchain' was originally coined by the pseudonymous Satoshi Nakamoto in their 2008 whitepaper describing the Bitcoin protocol. The majority of the research on blockchain and similar technologies references back to this seminal work. However, the underlying concept originates from Haber & Stornetta (1991) where they describe a system of 'linking' for timestamps that forms the basis for the blockchain concept.

The structure of a blockchain is described by Nofer et al. (2017). It is constructed of data containers called blocks which include in their metadata a cryptographic hash of the previous block, linking each block to it's predecessor in the chain. The presence of these hashes assures nodes that sufficiently aged blocks are computationally infeasible to modify without detection (As this would change the hash), and therefore can be treated as immutable.

3.1.1 Consensus

Public blockchains rely on a 'consensus mechanism' to ensure eventual consistency. This mechanism is used to resolve conflicts in the case that two nodes have diverged. Consensus mechanisms consist of two parts: an authorisation mechanism, and a conflict resolution strategy.

The authorisation mechanism serves to limit the ease with which individual nodes can submit new blocks to the chain. It should make it sufficiently difficult for a malicious node to produce significantly more valid blocks than an honest node, but ensure that it is computationally cheap for a node to verify a block's validity. The simplest example of this is proof-of-work, which requires the submitting node to find the solution to a hard problem. A large number of authorisation mechanisms exist including proof of work, stake, participation, authority and personhood.

The conflict resolution strategy determines how a node handles conflicts with the history stored on other nodes. As adding new blocks is rendered difficult by the authorisation mechanism, we can assume that the longest valid chain of blocks represents the truth. This is because the longest chain should theoretically have been produced by the largest group of nodes following the same procedure, and we assume that the majority of nodes within the network are honest.

3.1.1.1 Proof of Work

PoW is the original mechanism used to ensure consensus within the bitcoin blockchain and as such serves as the baseline for consensus algorithms. It uses a cost-function such as Hashcash (Back, 2002) to ensure that adding new data to the chain is associated with a significant cost. Using a PoW model means that a malicious actor must control 50% or more of the compute dedicated to the chain to be able to append invalid data.

PoW's inherent weakness is its cost. For proof of work to be an attractive prospect, there must be an incentive to do the work. As rewards decrease, or in their absence, there is no incentive for nodes to expend the resources required to generate valid proofs-of-work. This can lead to a loss of node variety and make the network vulnerable to Sybil attacks. Additionally, PoW has come under scrutiny for its sustainability. A 2018 analysis by de Vries highlighted the growing problem of Bitcoin's energy consumption, demonstrating that the energy requirements of the network were competitive with that of some smaller countries. This was further explored by Neumueller (2023) in a review of the methodology behind the Cambridge Bitcoin Electricity Consumption Index, which adjusted previous estimates down but still demonstrated extremely high energy demands.

3.1.1.2 Proof of Authority

Proof of Authority (PoA) is described by Joshi (2021) as a 'permissioned consensus protocol' that achieves consensus through 'authorized nodes'. In practice, this means that the eligible generators for a given block are limited to a fixed, publicly known, list of nodes known as sealers. This approach has significant advantages in terms of compute efficiency and throughput as we don't need to require nodes to do large amounts of redundant work to validate a block. Instead, blocks are simply signed by one of the trusted validators which can be trivially validated by other nodes.

PoA systems are inherently reliant on the diversity of the validator list. If the validator list lacks in diversity, the validators can collude to censor blocks or validate invalid blocks. For this reason, PoA is most commonly used within private blockchains where the integrity of nodes does not need to be questioned.

For PoA to be used in a public context, the contents of the validator list is generally based on reputation. This is sometimes referred to as 'Proof of Reputation' (Aluko & Kolonin, 2021). Different projects approach reputation differently, but the general consensus is that reputation should be hard to gain, but easy to lose if acting maliciously. Additionally, reputation functions should be monotonic such that it is always preferable to have a higher reputation score over a lower one.

3.1.1.3 Proof of Personhood

Proof of Personhood (PoP; Borge et. al, 2017) is a novel variant of PoA which leverages a list of verified humans as it's source of authority. This personhood comes with an inherent authority, and also provides strong incentives for acting honestly within the system. Strikes against an individuals reputation can be viewed almost as a strike against their personhood, and may be associated with other systems such as access to communal services and benefits.

The key issue within the development of PoP is in building a valid list of public keys with a 1:1 relationship with real people. Many projects have suggested ways to achieve this, but all approaches lack real world testing and validation. For the sake of this project we assume that we already have access to a valid census as this is also required to determine voter eligibility. As a result, despite it's incomplete definition, PoP can be an effective consensus mechanism within the system.

3.1.2 Sidechains and Rollups

Another useful concept within blockchains is that of a 'sidechain'. This is a secondary, independent, blockchain which has some predefined interaction with a main chain. A

common example of this is 'rollup' chains, which are used to process batches of transactions which are then submitted to the main chain in a more efficient, preprocessed form. The state of the art in sidechains was reviewed satisfactorily by Singh et al. (2020).

3.1.3 Limitations

Blockchain also suffers from a few key drawbacks. The most significant of these is the fact that blockchains continuously expand in size as they must store the entire chain history for validation. Additional weaknesses include only guaranteeing eventual consistency and having limited throughput.

3.2 Version Control

Version Control concerns the general concept of storing an ordered history of changes to a set of documents. Version control systems (VCS) such as Git and Subversion have been in use by programmers for years as the primary method for collaboration on codebases. Most VCS allow a user to summarise their changes into a 'diff'; a file containing the smallest set of addition and deletion operations required to move from the previous state to their current state. Generating the diff is done through algorithms which solve the longest common subsequence problem, such as that described by Myers (1986).

Version control is useful for this project as it provides a mechanism by which a participant can suggest changes to the shared documents, and the mechanism by which those changes are eventually ratified.

3.3 Peer to Peer Networking

A peer to peer networking approach underlies any convincingly decentralised system. A comprehensive review of relevant P2P networking concepts is given in the 'Handbook of Peer-to-Peer Networking' (Shen et al.; 2010). Peer to peer networks are formed amongst a group of equally potent peers, without any specific server to act as a trusted source of truth. As a result, it is important when building a peer to peer networked application that each node is capable of independently verifying the data it receives from the network.

In the context of this project, efficient peer discovery is an important feature as we expect to have a large number of ephemeral ballot-casting clients. One of the most common approaches to this within existing systems utilises random walks over a distributed hash table (DHT). For example, this approach is leveraged by the LibP2P ecosystem underlying the IPFS network (LibP2P; 2024) via the Kademlia DHT.

For a Peer to Peer network to be resilient it must also be sufficiently distributed, meaning that nodes have a sufficient level of connectedness to the overall network. Additionally, the network should be organised such that it avoids the formation of isolated clusters.

3.4 Homomorphic Encryption

Homomorphic encryption schemes provide a cryptographic structure where operations can be performed directly on the encrypted data. This allows the proposed system to perform calculations such as weighting and tallying without needing to decrypt and expose the underlying values.

Initial homomorphic schemes such as Paillier (Paillier, 1999) and ElGamal (ElGamal, 1985) are only capable of a limited operation set under encryption. For example, Paillier only

supports addition between two ciphertexts. In the context of voting, addition only is usually sufficient as votes only need to be weighted by a plaintext scalar or tallied.

More complex schemes, known as fully homomorphic encryption, have been proposed and implemented such as Zama's Fully Homomorphic Encryption over the Torus (Chillotti et al. 2021). These cryptosystems are able to support the complete range of operations on the underlying data including boolean and bitwise operations, however they are generally less computationally efficient.

3.5 Multi-Party Computation (MPC)

Another important concept within trustless decentralised networks is that of Multi-Party computation. This encompasses a class of algorithms which necessitate the cooperation of a number of independent nodes. This approach mitigates the risk posed by bad actors by spreading computation over a quorum of the network.

3.6 Threshold Encryption & Distributed Key Generation Schemes

Threshold encryption is a strategy for creating a ciphertext which can only be decrypted through the cooperation of a large number of (ideally independent) nodes via a Multi-Party Computation protocol. A threshold encryption scheme leverages a shared public key which is used by all clients to generate encrypted data packets. Threshold generalisations exist for a number of cryptosystems, such as Paillier (Veugen et al., 2019)

Threshold encryption protocols split the private key among a number of nodes as a series of private key shares, with some redundancy factor. This means that no individual node is capable of decrypting private information. When decryption is requested, nodes must cooperate by performing partial decryptions which can then be combined to produce a decrypted result.

Threshold key pairs can be generated through the cooperation of a large number of nodes in a Distributed Key Generation protocol, or through a trusted third party dealer. Obviously, a distributed protocol is ideal, such as the one described by Hamidi & Ghodosi (2023).

3.7 Vote Tallying Protocols

Tallying within voting systems poses a specialised problem space which combines a number of the problems solved above with some novel parameters. Primarily, the difficulty stems from simultaneously needing a tally which is verifiable and trustworthy, but which does not compromise the privacy of voters.

A survey by Cortier et al. (2021) provides a thorough summary of approaches to tally-hiding strategies including an overview of MPC, appropriate homomorphic encryption schemes and result functions.

A few complete protocols for vote tallying have been proposed. Of these, the Helios protocol (Adida, 2008) is the most prominent having been used for a number of real-world election use cases. A more recent proposal by Küsters et al. (2020) named Ordinos describes an improvement upon the Helios system which improves privacy by only revealing the result, not the tally values. This is done through a multi-party computation protocol for greater-than and equality tests described by Lipmaa and Toft (2013).

3.8 Commitment Schemes

Commitment schemes, as summarized by Goldreich (2001), are a way in which an individual can publicly commit to a value without revealing that value until a later stage. Commitment schemes are often based on one-way functions. In the case of voting schemes they are useful to allow a voter to submit a 'sealed', immutable, ballot without making the contents publicly viewable. Additionally, within the wider context of blockchain rollups we can use Merkle Trees as a commitment scheme, committing to a given chain state.

3.9 Zero Knowledge Proof

A zero-knowledge (ZK) proof, originally proposed by Goldwasser et al. (1985), is a protocol in which a prover convinces a verifier of a statement in a way which does not reveal the underlying data. A simple example is proving knowledge of a secret key without revealing the key itself, or proving that a committed secret value conforms to some set of constraints. ZK proofs are useful when there is a need to validate information in a compact manner without compromising the privacy of a user.

3.9.1 Proof Systems

Within the literature, many systems have emerged for building generic arithmetic circuits which can prove statements with zero-knowledge non-interactively. Bitansky et al.'s 2012 description of 'zero-knowledge succinct non-interactive arguments of knowledge' (zk-SNARKS) establishes a class of proof systems with five key properties:

- Zero-Knowledge – The proof does not reveal any private information to the verifier
- Succinctness – The proof is small, containing a constant number of elements. Additionally, the proof can be verified in sub-linear time
- Non-Interactive – The proof can be verified without interaction with the prover
- Argument of Knowledge – A malicious prover cannot produce a proof of a false statement which will be accepted by an honest verifier
- Completeness – If both parties are honest, the verifier will always accept the proof

The zk-SNARK constraints are implemented by a number of schemes, many of which include a 'trusted setup' step. This is usually a one-time step where parties agree on some public constant values, generally proving and verifying keys. This step represents a risk as private values are generated which can be used to produce false proofs. Generally, these values are computed using multi-party computation and are based on the trust that at least one participant correctly destroys their private variables.

An early example of trusted-setup based zk-SNARKs was the Pinnochio system (Parno et al., 2013). Trusted-setup based zk-SNARKS remain in use for the ZCash cryptocurrency (Teor, 2021). Systems such as Spartan (Setty, 2019) do not require a trusted setup step, but still conform to all of the constraints of the zk-SNARK scheme.

Other popular systems include zk-STARKs (Ben-Sasson et al. 2018) and Bulletproofs (Bünz et al. 2017). These systems provide similar functionality to a zk-SNARK system with different benefits and drawbacks in terms of proof size, algorithm complexity and underlying security assumptions.

	SNARKs	STARKs	Bulletproofs
Algorithmic complexity: prover	O(N * log(N))	O(N * polylog(N))	O(N * log(N))
Algorithmic complexity: verifier	~O(1)	O(polylog(N))	O(N)
Communication complexity (proof size)	~O(1)	O(polylog(N))	O(log(N))
- size estimate for 1 TX	Tx: 200 bytes, Key: 50 MB	45 kB	1.5 kb
- size estimate for 10.000 TX	Tx: 200 bytes, Key: 500 GB	135 kb	2.5 kb
Trusted setup required?	YES	NO	NO
Post-quantum secure	NO	YES	NO
Crypto assumptions	DLP + secure bilinear pairing	Collision resistant hashes	Discrete log

Figure 1. Comparison of ZK Proof System Features – Adapted from Matter Labs (2023)

3.9.2 Usages

zk Protocols can be used in a wide range of circumstances, often being used for proving properties of committed values. Examples of properties with established proof protocols include:

- Proof of Plaintext Knowledge
- Proof of Plaintext Correctness
- Proof of Homomorphic Operation Correctness
- Proof of Keypair Generation Correctness
- Proof of Group Membership
- Proof of Computation Correctness / Protocol Adherence
- Proof encrypted value decrypts to 0
- Proof encrypted value decrypts to value in range [0, q)

Within the context of voting systems, of these a few are of greater importance. Range proofs are useful for establishing bounds on cast ballots, preventing double casting. Group membership proofs can be used to prove a users right to vote, and homomorphic correctness proofs can be used to verify the integrity of tallies.

Chapter 4: System Design

4.1 Desirable Features

To effectively compare the suggested architecture with existing systems and evaluate design decisions it is useful to establish a set of desirable features. From the analysis of existing systems we can construct a set of factors which address known weaknesses and preserves existing strengths.

Firstly, a system implementing LD must support four features: Direct Democracy, Flexible Delegation, Meta-Delegation and Instant Recall. Therefore, we can evaluate any proposal or implementation on the basis of it's support for these four features.

Secondly, we observe that existing LD systems have generally been limited in their success due to a lack of trust or a lack of binding power. To resolve these, we propose three features:

1. Decentralisation – Processes and data storage are not controlled by any one trusted party. Security is assured through cryptography and computation rather than trust and secrecy. Any willing individual is able to participate in assuring the security of the system
2. Self-Tallying – The system is capable of autonomously calculating the result of votes
3. Self-Ratifying – A new property proposed by this paper. This outlines the ability of a system to action the results of a referendum into real changes in a system of regulations

Thirdly, we can leverage the work of Onur and Yurdakul (2022) who outlined a system of requirements for secure blockchain-based voting systems leveraging a wealth of research both on blockchain systems and within traditional systems. These requirements are:

- Eligibility
- Uniqueness
- Privacy
- Universal Anonymity
- Fairness
- Accuracy
- Universal-Verifiability
- Individual-Verifiability
- Robustness

These characteristics provide strong assurances of a systems trustworthiness and resistance to bad actors during voting processes.

Finally, we explicitly include the concept of 'coercion-resistance', described by Finogina and Herranz (2023). This may be considered to be largely covered under the privacy and universal anonymity features of secure B-voting. However, in the context of delegations, additional considerations may be required to ensure that votes are coercion resistant.

The full set of requirements, as well as the fittingness of systems discussed earlier, is visualised in the following.

System	Liquid Democracy				System Independence				Secure B-Voting								
	DD	FD	MD	IR	D	ST	SR	CR	E	U	P	UA	F	A	UV	IV	R
Liquid Feedback	Y	Y	Y	Y	N	Y	N	M	Y	Y	M	N	M	Y	N	B	M
Snapshot	Y	M	M	B	M	Y	N	B	Y	Y	M	N	Y	Y	M	N	Y
ElectAnon	Y	N	N	N	M	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Figure 2. Constraints-based comparison of existing solutions

Key: Y – Fully Supports, M – Mostly Supports, B – Barely Supports, N – No Support

DD: Direct Democracy, FD: Flexible Delegation, MD: Meta Delegation, IR: Instant Recall, D: Decentralisation, ST: Self-Tallying, SR: Self-Ratifying, CR: Coersion Resistant, E: Eligibility, U: Uniqueness, P: Privacy, UA: Universal Anonymity, F: Fairness, A: Accuracy, UV: Universal Verifiability, IV: Individual Verifiability, R: Robustness

4.2 Proposed Architecture

The first thing produced by this project is a proposed architecture for the overall system. This architecture provides the template against which we can assess the merits of the system.

The proposed architecture is comprised of 4 interconnected components:

1. A document storage blockchain – Responsible for storing the repository which is being democratically managed, and ensuring its integrity
2. A collection of voting rollup sidechains – Responsible for storing proposed changes and all of the votes cast towards that issue
3. A delegation storage blockchain – Responsible for representing active delegations
4. A census service – Responsible for maintaining a 1:1 relationship between public keys and valid human identities

We also include in the architecture diagram a virtual trustee component, which represents the responsibility of evaluating and validating votes before ratification. However, this component is deeply integrated with the document storage chain and in an implementation would not be cleanly separable. Finally, the architecture includes provision for client applications for accessing the system.

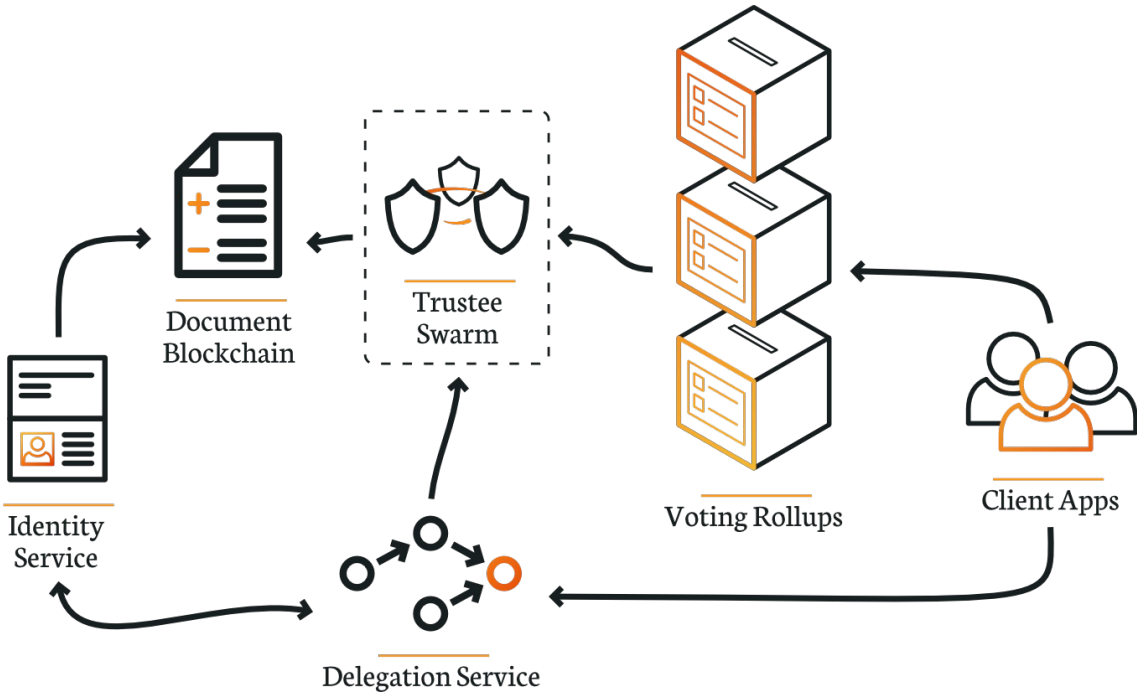

Figure 3: A high-level overview of the layout of the network components and their dependencies

It is intended that the majority of 'full' nodes will opt to run all components in parallel, which will significantly reduce network strain. However, potentially it may be worthwhile for nodes to be able to elect to only run a subset of the features.

4.3 Consensus Mechanism

Multiple elements within the system require a blockchain, and for this we require a consensus mechanism. We suggest that the most appropriate consensus mechanism is a proof-of-reputation system which leverages personhood as it's baseline. This provides strong incentives for compliant node behaviour and also works to make it prohibitive to amass the resources to stage a system attack – if an attacker has access to a sufficient body of support to act maliciously they also probably have sufficient support to simply pass votes correctly.

For a block to be considered valid under this proof-of-reputation system, it must have been validated by a node with sufficient reputation. For the purposes of this mechanism, we associate each node with a single identity from the census service. Each block produced is signed against this identity, and any penalties are also applied against that same identity. We also must include a proof of the signing node's reputation at the point of signing to prevent reversion attacks. If an identity signs an invalid block, it will have a reputation penalty applied in subsequent blocks. Eventually, if a node submits too many invalid blocks all blocks produced by that node will be deemed invalid by any honest nodes.

4.4 Document Store

A blockchain storing the version controlled document or set of documents. This chain is the core source of truth within the system, and constitutes the method by which the system achieves 'self-ratification'. Conceptually, the intention is that the community

leveraging the system would treat these documents as binding, and build appropriate enforcement systems which leverage the documents as a source of truth.

The blockchain enables version control by storing in each block a set of differentials. To obtain the most recent iteration of the document, each set of differentials can be sequentially applied to the original 'base' document. Through using a blockchain all modifications are immutable allowing a community to track when changes were enacted.

When signing a set of diffs into the document store blockchain, we include a reference to the voting chain which produced it alongside a set of ZK proofs of the correctness of the tallying protocol (As defined by the Ordinos system).

Within each block, we also store a number of hashes for validation purposes.

- The 'document hash' is the hash of the rendered document set. This allows us to verify if our document version, after applying all of the diffs up to this point in the chain, matches the version published by the block signer
- The 'diff hash' is the hash of the included diffs, and allows extra assurance that the diffs have not been modified
- The 'vote capstone' is the hash of the sealing block on the voting rollup chain which produced this block, providing a concrete reference to the exact version of the chain which was evaluated during the tally

Blocks are structured as follows:

```
Block {
    metadata: {
        document_hash: Hash,
        diff_hash: Hash,
        source_vote: Hash,
        vote_capstone: Hash,
        evaluation_proof: { ... },
        timestamp: DateTime,
    },
    diffs: { ... }
}
```

Figure 4: Document Chain Block Data Structure

For simplicity we assume that each block contains a full set of changes. If a change set would cause a block to become larger than the block limit it is up to the user to split that into a set of distinct semantically complete changes. As submitted changes will be mostly text, and we can allow a generous block size for the document chain, it is unlikely for this to ever pose a significant issue.

A useful feature of this design is that nodes do not need to store the full set of votes for each change, as they are stored off-chain in rollups. This reduces the risk posed by blockchain growth over time and should allow the document chain to be widely replicated on low cost nodes.

All of the data on the document chain is 'in the clear', and so if an inconsistency is later found (For example a suggested change led to an unintended side effect) it is sufficient for the chain to 'fail forwards'; Users can simply create a new vote with changes that fix or supersede the invalid ones.

4.4.1 Trustee Role

Additionally, the document storage nodes will perform the role of trustees who are capable of evaluating the result of votes. This is because the document chain will be the most widely distributed system and therefore the most resilient to corruption and distributed attacks. The document store is statically linked to a delegation and census service, to provide a consistent view of the voter base across votes and prevent preferential selection of delegations.

4.4.1.1 Distributed Key Generation

The first role of the trustees is to periodically perform a distributed key generation protocol. This provides the keys which are used on vote chains to encrypt vote intentions. Shares are distributed amongst the trustees with a threshold t required to collaborate to perform any decryption operations. A standard threshold is $n/3$, where n is the total number of participating nodes. This means that the swarm can lose $2t$ trustees without issue, and that more than 1/3 of nodes must collaborate to misbehave.

The key produced through this protocol should be regularly replaced, with participating nodes retaining a short history of keys to enable unsealing of maturing votes. How long private key shares are retained for is a tuneable security parameter, but should be set slightly higher than the expected time to maturity for most votes.

4.4.1.2 Vote Evaluation Protocol

Another key role of the trustee responsibility is to evaluate mature votes. This involves validating the set of changes and all of the cast ballots, homomorphically weighting and aggregating them, running a MPC protocol for to generate a verdict and enshrining successful changes onto the document chain. When a vote chain is mature, it is elected by the trustee swarm for evaluation. Vote evaluation selection, like many other actions within the system, is done at the risk of reputational score, preventing DoS attacks.

The vote evaluation protocol is characterised by the following steps:

1. Preliminary validations – Validate that the provided chain contains a valid set of changes and metadata
2. Chain Sync – Locally sync the chain to each trustee-capable node participating in the unsealing process. For changes which are ratifiable, it is likely that the chain will already have been synced to a large number of trustee nodes already during the casting process
3. Weighting – Taking a list of all individuals who cast a ballot towards the vote, we can process the delegation graph to generate weightings for them. Each ballot can be homomorphically multiplied by its derived weight to get a set of power-adjusted ballots
4. Evaluation – The trustees are now in a state where they can execute the Ordinos evaluation protocol. The result of this is a boolean verdict, alongside a Zero-Knowledge proof of correctness for the tally
5. Ratification – If the vote passes, the change can be 'ratified'. We create a new block on the document chain which includes both the changes (Sourced from the rollup) and the NIZKP of correctness for the evaluation process

4.5 Vote Rollups

Open decisions are represented by a collection of ephemeral 'Vote Rollup' (VR) chains, one per suggested change. VRs store proposed changes off-chain to enable higher storage efficiency and to reduce the risk of DoS attacks. They are responsible for storing the contents of the change itself alongside metadata such as its target document storage chain and a record of all cast ballots.

4.5.1 Ballot Structure

The core data structure within vote rollups is the ballots cast against an issue. Their construction must be carefully considered to enable verifiability and privacy whilst being compatible with the delegation and tallying components. To achieve this, ballots are characterized by the following structure:

```
Ballot {
    timestamp: OffsetDateTime,
    issue_id: String,
    vote: Ciphertext<Vec<u64>>,
    range_proof: ZKRangeProof
}
```

Figure 5: Ballot Data Structure

We encode votes as a vector with two elements, one representing the for vote and one representing the against. This enables us to homomorphically aggregate each field, and determine which of the two is greater using a 'greater-than' MPC protocol, such as the one described by Lipmaa and Toft (2013), to evaluate a result. We also include a corresponding ZK Range proof that the encoded values are either 1 or 0, preventing double casting.

The timestamp and issue_id fields are included for verification purposes. Additionally, each ballot cast is sent alongside a signature. This signature is primarily used to enable the trustees to link ballots to a user's delegation. However, this also enables individual verifiability of vote-cast correctness as it ensures that the vote on-chain was produced by the correct caster.

4.5.2 Life Cycle

A voting sidechain moves through a series of stages over its lifetime:

Figure 6: Vote Lifecycle

4.5.2.1 Genesis

For each set of proposed changes, a brand new VR is created. The first block on the chain establishes:

- The contents and intentions of the suggested changes
- Which document chain should evaluate the final result
- Which public key ballots should be encrypted against
- What conditions trigger evaluation, for example a specific point in time or a quantity of votes

Genesis blocks are structured as follows:

```
GenesisBlock {
    name: String,
    description: String,
    metadata: {
        target_store: DocChainRef,
        timestamp: DateTime,
        eon_key: PublicKey
    },
    diffs: { ... }
}
```

Figure 7: Genesis Block Data Structure

4.5.2.2 Ballot Casting and Propagation

Once the initial set of changes have been proposed, the VR will become 'active' and open to clients casting ballots. Ballots are pooled to be included in the next available block and propagated to other nodes in the network via a publisher-subscriber messaging queue. Blocks during this phase are constructed whenever a sufficient number of ballots are available in the pool, and just contain an array of ballots.

When a node receives or propagates a ballot for a vote, that node will evaluate if it should start tracking that VR itself. This evaluation considers the maturity of the VR, whether it is stored with its immediate neighbours and the available space on the node. The exact criteria may be left open to node operators to allow them to tune their replication level in line with their available hardware and storage. In effect, this means that as a VR draws closer to being ratifiable it naturally becomes more widely replicated.

4.5.2.3 Evaluation

Eventually, a vote will meet one or more of the conditions for evaluation. At this stage, the evaluation process is initiated with the document chain by one or many nodes. Whilst the document chain is performing the evaluation, changes to the vote chain should be frozen. New ballots are collected in the pool but not propagated. Once the evaluation is completed, if successful, the VR should generate a sealing block including the proofs of correctness for the evaluation process retrieved from the document chain. At this point any unprocessed ballot submissions can be discarded.

4.5.2.4 Challenge Period

Immediately after ratification, a change enters a challenge period. At this stage, some document chain nodes become keepers of a given vote chain. This prevents the results of the vote from being forgotten during the challenge period. Data integrity is ensured by storing the hash of the sealing block on the document chain.

During this time, nodes can evaluate the stored proofs of correctness as well as data on-chain to verify that the voting procedure was executed correctly. If a node identifies an issue, it can broadcast the error to the rest of the network. If the rest of the network finds that the change submitted was invalid then the erroneous block can be rolled back and the appropriate penalties applied to the misbehaving identities.

4.5.2.5 Pruning

After a predetermined challenge period the contents of the VR can be discarded. This should help greatly in terms of storage efficiency as the overwhelming majority of data stored across the various chains will be ballot blocks.

4.6 Delegation Store

The delegation layer stores a versioned graph representation of active delegations. Each time a delegation is changed, the store persists the new delegations. Changes to delegations are made by submitting a signed copy of the intended delegate's public key using their own private key. This ensures that only the delegator is able to change their own delegation.

The delegation store stores a map of *delegator* to *delegate* public keys. This ensures that there is always a 1:1 relationship between delegator and delegate. The map representation can also be conceptualised as a directed graph, where simple graph traversal algorithms to calculate weights. The map structure does not enforce the resultant graph to be acyclic, but does ensure that it is traversable provided that we keep track of already visited nodes.

The service is proposed to be a blockchain of its own, with each block storing a set of updated delegations. To get the current delegation map, we simply pull the most recent signed delegation modification from this chain for each public key. We establish the delegation service as a component separated from a specific document or vote chain to enable individuals to have their delegations seamlessly mirrored across multiple domains, without requiring additional user action to achieve this.

4.7 Census Service

The identity service is responsible for validating whether a provided public key is associated with a valid human identity. It is also responsible for ensuring a strict 1:1 relationship between valid public keys and key owners.

Unfortunately, no approaches found in the literature or during the course of this project provide a satisfying level of robustness. This project leaves this problem open to future research. For the sake of testing out the proof-of-concept, we assume that each node simply has access to a database of public keys which it can reference as it's census without approaching the problem of synchronisation.

Chapter 5: Implementation

Alongside the proposed system architecture, we provide prototype implementations of a few features of the system. The proof-of-concept components were developed using all of the same technologies as would be expected for a real implementation of such a system. This allowed a better understanding of exactly what work will need to be done for a widely usable implementation.

5.1 Technical Decisions

5.1.1 Rust

Rust was selected as the implementation language of choice for this project because it offers low-level speed, robust memory safety and is a commonly used language within the subject area. Specifically, the quantity and quality of libraries available implementing peer to peer networking, homomorphic encryption, zero knowledge proof and other key technical elements of the project was deemed to be superior within the Rust ecosystem over many others.

Rust does come with unfortunate overhead in terms of language complexity however, which may have unnecessarily slowed implementation for prototyping purposes where implementation robustness may not necessitate such strict guarantees.

5.1.2 LMDB

Much consideration was made into selecting an appropriate key-value store implementation. Options such as RocksDB, Sled and SQLite were all evaluated. In the end, LMDB was selected as a battle-tested option which offers best in class speed and resilience. It offers all of the functionality required: ACID compliance, ordered storage, cheap read transactions, deadlock free writes and multithread support. LMDB is used for data storage within the Monero project which is very mature and has similar requirements to this project, lending additional credence to its selection.

5.1.3 Paillier Encryption

For the homomorphic encryption scheme we selected 'Paillier' encryption (Paillier, 1999). This provides a well understood and tested scheme, which has established precedents for threshold decryption (Fouque et al., 2000), distributed key share generation (Hamidi & Ghodos, 2023), and key zero-knowledge proof protocols such as range proofs (Boudot, 2000).

5.2 Components

The proof of concept implementation focused on implementing strictly the components necessary to explore the voting protocol. This included implementing the data structures required for representing delegations, establishing a baseline implementation for the vote rollup blockchains, implementing ballots and some of their validation mechanisms and building an initial version of the vote execution phase. All in all, the proof of concept software comprises around 2000 original lines of code, as well as a few modified versions of downstream libraries.

The implementation of this is split into two higher order components.

5.2.1 Node Service

The node service is the bulk of the implementation. It includes a barebones blockchain implementation for representing vote rollups, components for in-memory representations of a census and delegation graph, and a simplified implementation of the trustee behaviours exemplifying how delegations are applied.

This service runs as a LibP2P server node. It supports node discovery and identification via the mDNS, Kademlia and Identify Protocols. It is able to receive cast ballots over a GossipSub messaging queue and process those into a pool. Once the node meets the conditions for block creation, it is able to pull ballots from the pool, assemble a block and submit that to the chain. The chain also includes a synchronisation protocol which allows nodes to check with neighbours to update to the longest version of the chain.

Chain data is stored locally using the heed library for accessing a LMDB store. ed25519 elliptic curve cryptography is used for digital signatures, Paillier encryption is used for the homomorphic scheme. Data is sent over the wire and stored on disk using compact binary encoding schemes.

5.2.2 Client CLI

The provided client CLI represents a prototype of the actions that an end user can perform within the network. In the implementation provided, this includes establishing a new identity via the generation of a signing keypair and the ability to cast a vote. The vote casting protocol constructs a ballot packet by generating Paillier-encrypted ballots alongside ZK proofs of their correctness. These are then submitted to the network by spinning up an ephemeral LibP2P client node which submits the ballots to the topic and awaits confirmation of their receipt from one or more peers.

5.2.3 Shared Types

For simplicity a third crate is included which stores the types which are shared by the client and nodes, such as the structure of ballots.

Chapter 6: Results

6.1 Constraints Analysis

With the system architecture and makeup defined, we can evaluate the system against the constraint system established earlier.

6.1.1 Liquid Democracy

The system clearly meets all of the requirements for liquid democracy. Direct voting is assured through the inclusion of all census individuals as casters on the vote rollups. We allow for the delegation component via the delegation graph component and through the implementation of the graph traversal algorithm we also enable meta-delegation.

Instant recall is mostly satisfied with the only caveat being that we rely on eventual, not immediate, consistency. This means that recall is assured only after a delay in line with the consistency model. However, as individuals are also able to directly vote to override in the short-term this drawback can be considered to be satisfactorily mitigated.

6.1.2 System Independence

The proposed system architecture establishes a pattern allowing for a completely decentralised system. Through the usage of blockchains and the proof-of-reputation consensus mechanism there is a clear pathway to constructing each component in a decentralised manner. The only limitation in this regard is the established lack of a compelling precedent for identity verification in a decentralised manner. Whilst the use of a blockchain allows us to store this list in a decentralised way, further research will be required to establish mechanisms by which it can be effectively managed.

The system leverages the work done by Ordinos to fit the self-tallying constraint, and the addition of a document store extends this self tallying property to conform to the needs of self-ratification.

Finally, the coercion resistance constraint is mostly met. Due to the delegation components, we have to reveal the chains of delegation, and the list of those who have voted on a given topic in the clear. This potentially opens up an attack where an individual is coerced into a specific delegation, and also is coerced to not manually cast their vote. Future work may be able to find a fix for this through the development of a more complex multi-party computation protocol allowing weighting without plaintext reveal of the delegations. However no such solution was found during this project.

Fortunately, such an attack is likely to not be scalable due to the way that power is inherently decentralised by leveraging a liquid democratic system. For an individual to influence a vote in this way would require coercion of enough people that a dedicated attacker would be in a position to simply perform a sybil attack on the network.

6.1.3 Secure Blockchain Voting

The constraints of the secure-blockchain model established by Onur and Yurdakul (2022) are satisfied as follows:

- Eligibility – Assured by the inclusion of the identity service which can be used to verify eligibility

- Uniqueness – Assured by storing votes with a signature on the votechain and taking the most recent at tally-time
- Privacy – Assured by encrypting votes on-chain against a threshold key
- Universal Anonymity – Partially assured by implementing a system which is isolated from any financial activities. However, a voters activity in voting is linkable to their identity
- Fairness – Ensured by leveraging the Ordinos protocol and preventing further vote modifications by applying a capstone block.
- Accuracy – Enabled through the use of homomorphic encryption allowing deterministic calculations and assured through the use of NIZKPs and reputation penalties for bad action
- Universal-Verifiability – Assured by including a series of NIZKPs of the correctness of the MPC protocol execution
- Individual-Verifiability – Enabled by the client apps and through storing all vote data in public on a blockchain
- Robustness – Assured by leveraging bad-actor resistant MPC protocols

The final system can be compared with the existing solutions in the following table:

System	Liquid Democracy				System Independence				Secure B-Voting								
	DD	FD	MD	IR	D	ST	SR	CR	E	U	P	UA	F	A	UV	IV	R
Liquid Feedback	Y	Y	Y	Y	N	Y	N	M	Y	Y	M	N	M	Y	N	B	M
Snapshot	Y	M	M	B	M	Y	N	B	Y	Y	M	N	Y	Y	M	N	Y
ElectAnon	Y	N	N	N	M	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Pnyx	Y	Y	Y	Y	Y	Y	Y	M	Y	Y	Y	M	Y	Y	Y	Y	Y

Figure 8. Constraints-based comparison of existing solutions with Pnyx

Key: Y – Fully Supports, M – Mostly Supports, B – Barely Supports, N – No Support

DD: Direct Democracy, FD: Flexible Delegation, MD: Meta Delegation, IR: Instant Recall, D: Decentralisation, ST: Self-Tallying, SR: Self-Ratifying, CR: Coersion Resistant, E: Eligibility, U: Uniqueness, P: Privacy, UA: Universal Anonymity, F: Fairness, A: Accuracy, UV: Universal Verifiability, IV: Individual Verifiability, R: Robustness

6.2 Prototype Testing

Alongside evaluating the model suggested, we also must evaluate the results of the proof of concept implementation. To examine it, a number of test scenarios were produced to verify correctness.

6.2.1 Unit Tests

Firstly, key individual components and processes were unit tested. The ballot structure is unit tested by building a test ballot packet and verifying it's correctness, specifically verifying that the range proofs within are correctly formed and valid. We also verify that the signatures produced are well formed and validate correctly.

We also include tests for blockchain validation on the votechain, verifying that blocks and the ballots within them both correctly validate, and reject invalid blocks. Alongside this there are tests for census properties and delegation resolution.

6.2.2 Delegation Testing

One of the key prototype components was delegation. To test this component, we can devise a number of graphs and assert that the code correctly calculates their weights. These scenarios aim to establish that the delegation algorithms effectively handle:

- Simple Graphs
- Complex Chains of Delegation
- Cycles

The following four test simple test scenarios were devised. In each scenario, we have a delegation graph layout, and a set of individuals who have cast votes. We can then manually calculate an expected set of resultant weights which are verified in the test scenario.

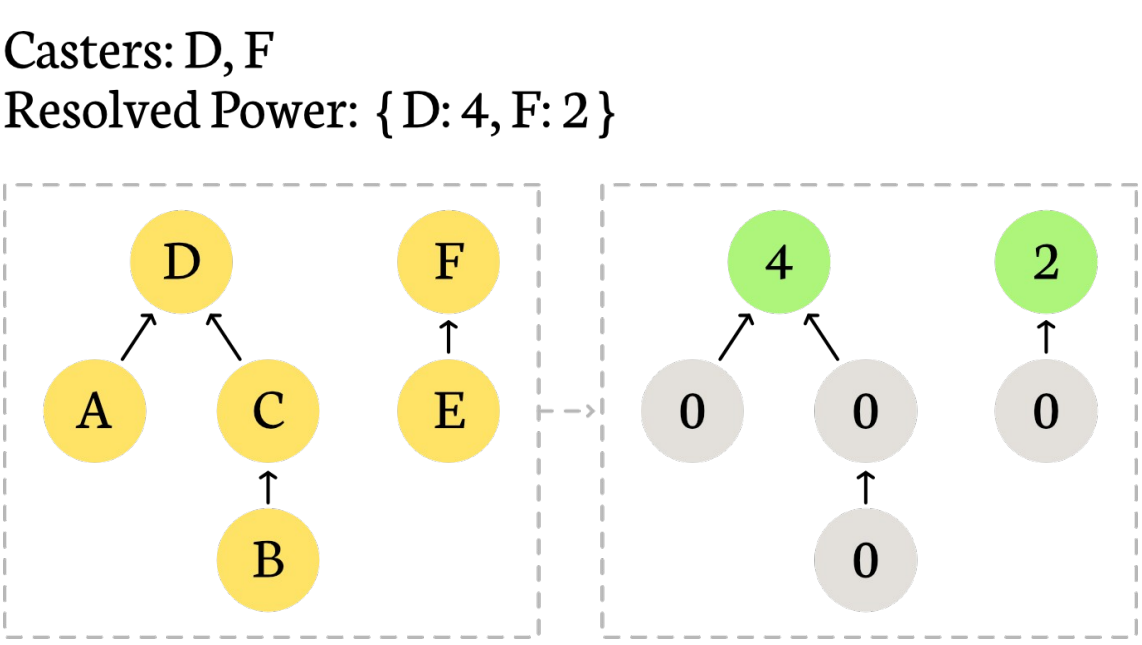

Figure 9. Simple Delegation Scenario

Casters: D, B, F
Resolved Power: { B: 1, D: 3, F: 2 }

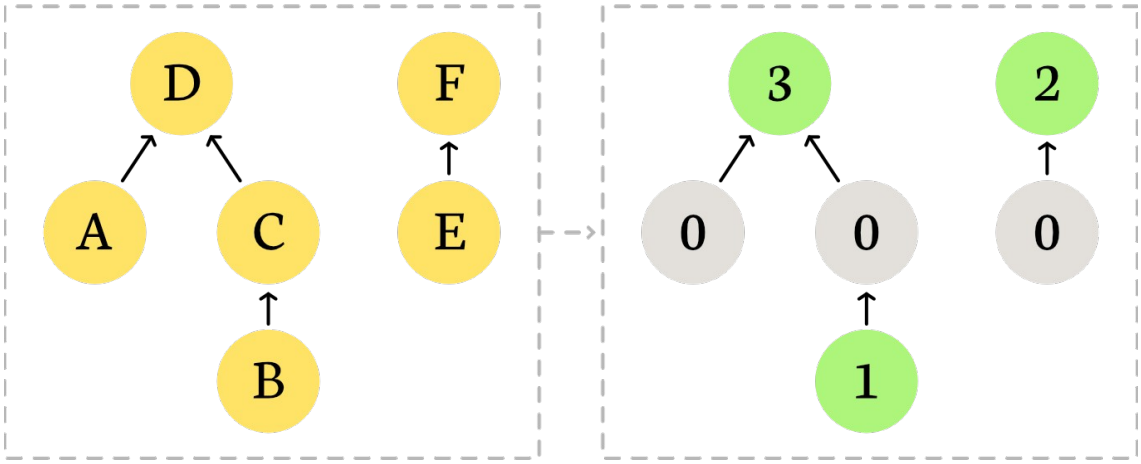

Figure 10. Delegation Scenario with cut off dependencies

Casters: A, B, C, D, E, F
Resolved Power: { A: 1, B: 1, C: 1, D: 1, E: 1, F: 1 }

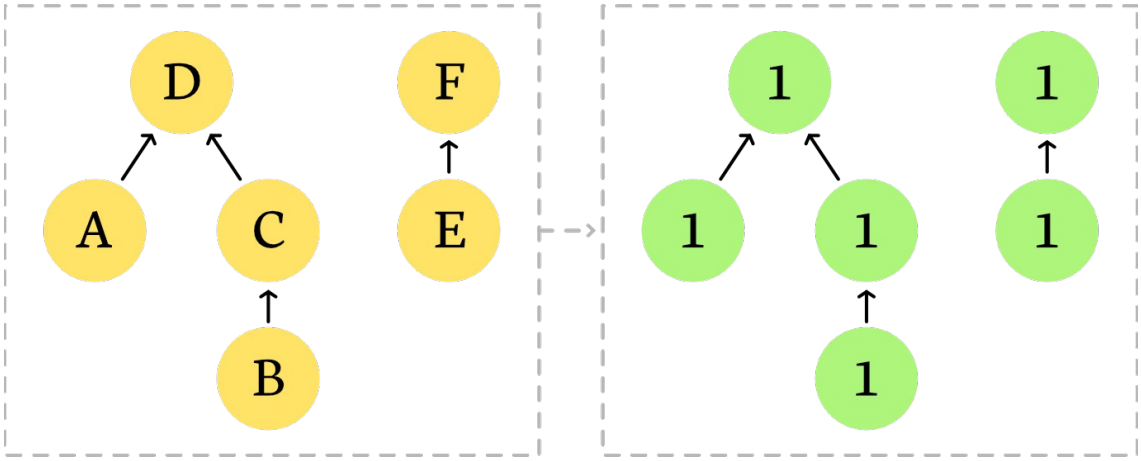

Figure 11. Delegation Scenario with Complete Voting

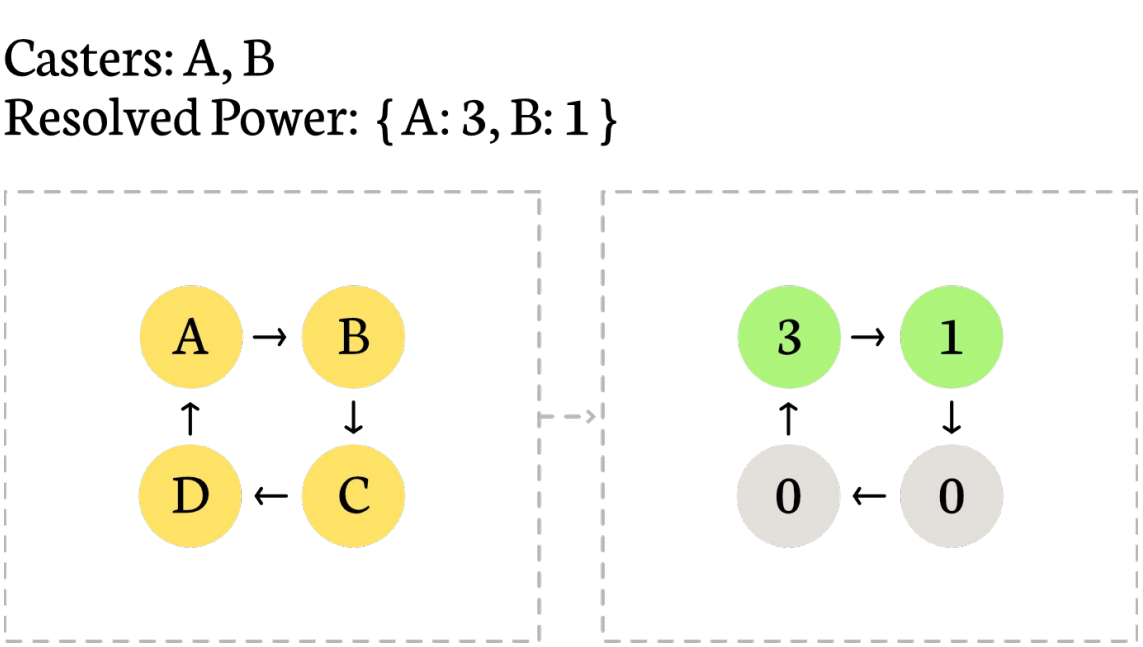

Figure 12. Delegation Scenario with Cyclic Dependencies

Running the tests, we can see that the delegation graph correctly calculates the resultant weights for each of these scenarios.

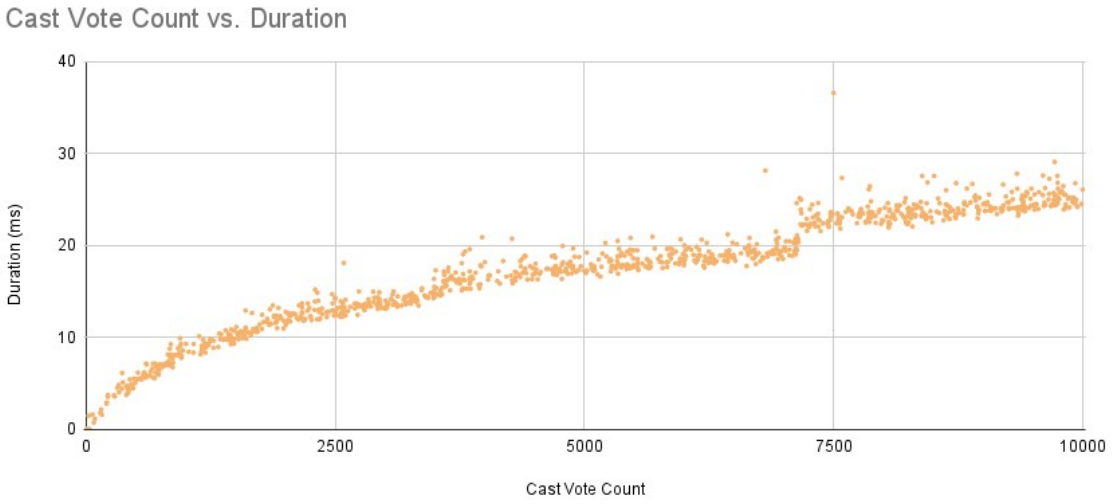

Figure 13. Delegation resolution latency against number of cast votes (10000 delegates)

Alongside the basic test scenarios, performance and stability were also tested by generating random delegation maps. 1000 test scenarios were run, each over a census with 10000 simulated individuals. A random set of votes was generated for each scenario and then the delegation graph was called to generate a list of weightings.

6.2.3 Swarm Test Scenario

The final test involves spinning up a test swarm and running an example vote. To do this, we spin up 5 nodes, verify that they correctly connect to each other forming a mesh

network, and then begin sending votes via the vote client. Once we have sent sufficient votes, we trigger a vote resolution calculation and generate a final result.

The following graph outlines the basic growth characteristics of the vote chain over a simplified test scenario. In this example, blocks are set to have a fixed size of two ballots and the evaluation is set to execute after the 5th block is included. Votes were cast using the client CLI against a set of pre-generated identities (Also created with the CLI)

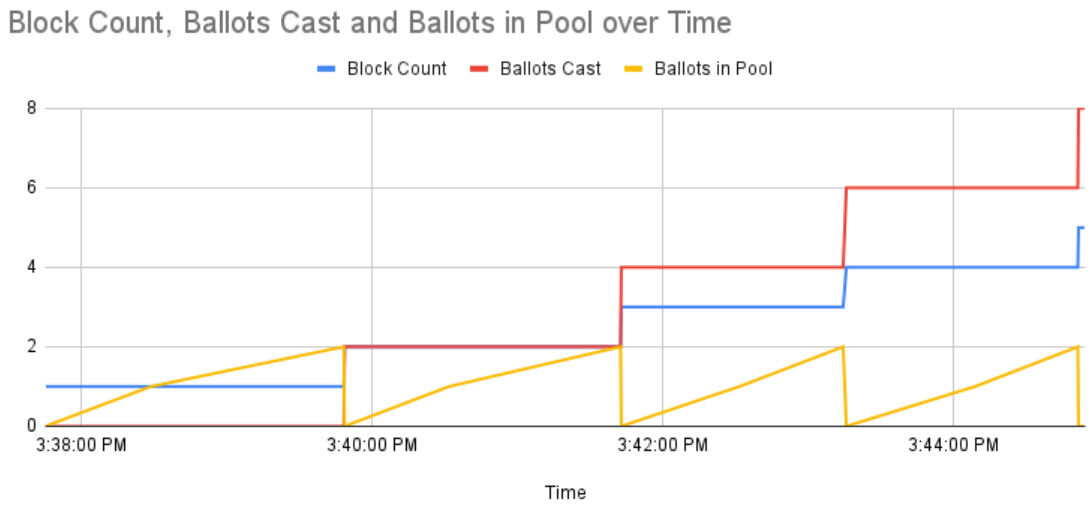

Figure 14. VoteChain characteristics through a simulation

6.3 Exhausted Research

On top of establishing a reasonable path forwards for implementing the proposed system, it is also worth highlighting a number of approaches which did not turn out to be useful.

6.3.1 MixNets and Shuffling

A commonly used approach for anonymising data in decentralised system is that of 'MixNets' or shuffling. This involve taking data packets and randomly reordering them such that their origin is no longer distinguishable. These have been applied in voting systems such as OpenVerificatum where ballots can be stripped of identifiers for mixing. However, in this system ballots must be held alongside a record of their caster for weighting in line with delegations.

6.3.2 Fully Homomorphic Encryption Schemes

We also considered fully homomorphic encryption (FHE) schemes, allowing for all operations such as division, boolean and bitwise (For example Zama's TFHE implementation, 2022). Whilst the capabilities of these can occasionally make certain constructions easier to define (For example certain zero knowledge proofs are more easily generalised under a FHE scheme), they are significantly less computationally efficient. This becomes especially relevant as we are going to be duplicating operations over a large number of nodes through multi-party computation protocols. As it seems that sufficient precedent exists for the relevant zero knowledge proofs over more simple systems, FHE was deemed unnecessary.

6.4 Known Limitations

Whilst the proposed solutions here aim to be as comprehensive as possible in light of the current state of the art, there are a number of key compromises or unknowns which may represent promising areas for future research. A few of these are:

6.4.1 Conflict Resolution

It is likely that, under the system suggested, a significant period of time may pass between a vote being opened and that vote resolving. In that time period, it is possible for other changes to be ratified which may make the contents of the suggested change no longer valid, or subject to misinterpretation. For the most common use cases this is unlikely, but there is space for additional research into more context-aware conflict resolution methods. For example, it is possible that different conflict-free replicated data types could be leveraged, or that AI models could be used for semantic conflict resolution.

6.4.2 DKG Implementation

During the development process we discovered that, whilst adequate descriptions exist, there is no ready-to use implementation of distributed key generation for Paillier encryption in rust. Of the protocols reviewed, the suggestion by Hamidi & Ghodosi (2023) seems to be suitable for this application. An efficient and side-channel hardened version of this protocol would be a large step forwards towards enabling the Pnyx system to be applied to real use cases.

Chapter 7: Future Work

7.1 Eager Ratification

In an ideal world, the results of a given voting chain could be periodically assessed to allow early ratification for popular votes rather than always waiting for a fixed timeout. This could be done, for example, by calculating a 'cast weight' metric, and evaluating if the total weight of votes cast is enough to potentially lead to a conclusive result.

This approach was not implemented within this project as it was not obvious how to enable this whilst both keeping the computational load on the trustees reasonable and not opening up a side-channel attack. Specifically, it may be possible for an attacker to tactically cast null or meaningless votes in a pattern which would trigger an evaluation attempt with only a single 'real' vote change between the previous and the attack. If this new evaluation resolves, then an attacker would be able to establish the nature of the vote cast by the last caster.

7.2 Participation Incentives

This specification does not ingrain any specific incentives for those running ratifying nodes, despite the risk they take to their reputational score. It is suggested that in any rolled out instance of this system, a proportional incentive would be provided for node operators. This could be increased voting weight, or benefits specific to the community they are participating in the governance of (For example a share of financial rewards). Establishing the exact nature of this, and how it would be delivered by the system, is another opportunity for further work.

7.3 Sociological Study

A huge opportunity for future work is available in the form of deeper study into how the system would be received and used by a real community. As discussed, Liquid Democracy is still an emergent field of research. Wide scale studies on the systems effect within a community, especially with the modifications focused on by this project, would be valuable to guide future developments.

7.4 Discoverability

The proposed system does not explore the mechanism by which suggested changes can be transmitted or discovered organically. This means that only massively popular and highly discussed changes will have any chance to get passed.

A dedicated mechanism – something like a bulletin board or forum – could be useful for increasing the visibility of less topical changes. This would be buildable as an additional frontend system leveraging data sourced from Pnyx.

7.5 Atomic Identity Management

As mentioned in the specification for the census service, the problem of identity management is left to future research. In current decentralised systems, anyone can create a new identity at any time. This is counterproductive for a voting system, as one person can simply create infinite personas to override the vote.

A potential solution to this is to have personas 'signed' by a number of individuals. This could include parents, doctors etc. The signing of new individuals would be publicly visible such that analysis tools could be built and used identify discrepancies in the system. Small quantities of duplication may be acceptable. Identities could be nullified by vote also.

There is a veritable swamp of ethical, technical and social issues to contend with within this area of research. An appropriate resolution to this problem is likely the most significant outstanding issue towards a fully satisfactory implementation of the proposed system.

Chapter 8: Conclusion

This project presents a major step towards truly democratic, long-term resilient governance and management of systematically important open source software projects. The proposed architecture optimises for a more comprehensive set of constraints than any of the existing systems explored. Additionally, it provides a meaningful improvement to usability and long-term viability through the inclusion of the self-ratification functionality. We have established that strong candidate technologies exist for the majority of critical functions in the system, proving that building and deploying a system conforming to the established constraints is technically feasible. We have also built a proof-of-concept implementation of a subset of the proposed components. This demonstrates concretely the viability of propagating, storing and evaluating votes in a liquid democratic context leveraging the discussed technologies.

Overall we posit that, with some of the stipulated future work completed, a full implementation of the system is both feasible and reasonable, and represents a compelling novel approach to the governance of open source, and more general, communities.

References

Anderson, T. (2023). More Rust ructions as project team confesses failure of “leadership chat”. https://devclass.com/2023/05/31/more-rust-ructions-as-project-team-confesses-failure-of-leadership-chat/

Turner, S. J. (2023). Why I left rust. https://www.sophiajt.com/why-i-left-rust/

OpenTofu. (2023). The OpenTofu Manifesto. https://opentofu.org/manifesto/

Colannino, J. (2021). What’s up with these new not-open source licenses? https://github.blog/2021-03-18-whats-up-with-these-new-not-open-source-licenses/

Mensching, R. (2024). A Microcosm of the interactions in Open Source projects. https://robmensching.com/blog/posts/2024/03/30/a-microcosm-of-the-interactions-in-open-source-projects/

HM Government. (2018). The Costs of the 2015 UK Parliamentary General Election.

Vassil, K., Solvak, M., Vinkel, P., Trechsel, A. H., & Alvarez, R. M. (2016). The diffusion of internet voting. Usage patterns of internet voting in Estonia between 2005 and 2015. Government Information Quarterly.

Springall, D., Finkenauer, T., Durumeric, Z., Kitcat, J., Hursti, H., MacAlpine, M., & Halderman, J. A. (2014, November). Security analysis of the Estonian internet voting system. Proceedings of the 21st ACM Conference on Computer and Communications Security.

Snapshot Labs. (2023). Snapshot Docs. https://docs.snapshot.org/introduction

Onur, C., & Yurdakul, A. (2022). ElectAnon: A Blockchain-Based, Anonymous, Robust and Scalable Ranked-Choice Voting Protocol. Distributed Ledger Technologies, 2(3). https://doi.org/10.1145/3598302

Paulin, A. (2020). An Overview of Ten Years of Liquid Democracy Research. The 21st Annual International Conference on Digital Government Research, 116–121. https://doi.org/10.1145/3396956.3396963

Blum, C., & Zuber, C. I. (2016). Liquid Democracy: Potentials, Problems, and Perspectives. Journal of Political Philosophy, 24(2), 162–182. https://doi.org/10.1111/jopp.12065

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Decentralized Business Review.

Haber, S., & Stornetta, W. S. (1991). How to time-stamp a digital document. Journal of Cryptology, 3(2), 99–111. https://doi.org/10.1007/BF00196791

Nofer, M., Gomber, P., Hinz, O., & Schiereck, D. (2017). Blockchain. Business Information Systems Engineering, 59(3), 183–187. https://doi.org/10.1007/s12599-017-0467-3

Back, A. (1997). Hashcash - A Denial of Service Counter-Measure.

de Vries, A. (2018). Bitcoin’s Growing Energy Problem. Joule, 2(5), 801–805. https://doi.org/10.1016/j.joule.2018.04.016

Neumueller, A. (2023). Bitcoin electricity consumption: an improved assessment. https://www.jbs.cam.ac.uk/2023/bitcoin-electricity-consumption/

Joshi, S. (2021). Feasibility of Proof of Authority as a Consensus Protocol Model. https://doi.org/10.48550/arxiv.2109.02480

Aluko, O., & Kolonin, A. (2021). Proof-of-Reputation: An Alternative Consensus Mechanism for Blockchain Systems. International Journal of Network Security & Its Applications, 13(04), 23–40. https://doi.org/10.5121/ijnsa.2021.13403

Borge, M., Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., & Ford, B. (2017). Proof-of-Personhood: Redemocratizing Permissionless Cryptocurrencies. 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), 23–26. https://doi.org/10.1109/EuroSPW.2017.46

Singh, A., Click, K., Parizi, R. M., Zhang, Q., Dehghantanha, A., & Choo, K.-K. R. (2020). Sidechain technologies in blockchain networks: An examination and state-of-the-art review. Journal of Network and Computer Applications, 149, 102471. https://doi.org/10.1016/j.jnca.2019.102471

Myers, E. W. (1986). An O(ND) Difference Algorithm and Its Variations. Algorithmica, 1(1–4), 251–266. https://doi.org/10.1007/BF01840446

Shen, X., Yu, H., Buford, J., & Akon, M. (2010). Handbook of Peer-to-Peer Networking. Springer US. https://books.google.co.uk/books?id=nXk_AAAAQBAJ

ProtocolLabs. (2024). LibP2P Documentation. https://docs.libp2p.io/

Paillier, P. (1999). Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In J. Stern (Ed.), Advances in Cryptology — EUROCRYPT ’99 (pp. 223–238). Springer Berlin Heidelberg.

Elgamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory, 31(4), 469–472. https://doi.org/10.1109/TIT.1985.1057074

Chillotti, I., Joye, M., & Paillier, P. (2021). Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks. https://doi.org/10.1007/978-3-030-78086-9_1

Veugen, T., Attema, T., & Spini, G. (2019). An implementation of the Paillier crypto system with threshold decryption without a trusted dealer. https://eprint.iacr.org/2019/1136

Hamidi, A., & Ghodosi, H. (2023). Efficient Distributed Keys Generation of Threshold Paillier Cryptosystem. In G. Bella, M. Doinea, & H. Janicke (Eds.), Innovative Security Solutions for Information Technology and Communications (pp. 117–132). Springer Nature Switzerland.

Cortier, V., Gaudry, P., & Yang, Q. (2021). A toolbox for verifiable tally-hiding e-voting systems. https://eprint.iacr.org/2021/491

Adida, B. (2008). Helios: Web-based Open-Audit Voting.

Küsters, R., Liedtke, J., Müller, J., Rausch, D., & Vogt, A. (2020). Ordinos: A Verifiable Tally-Hiding E-Voting System. 2020 IEEE European Symposium on Security and Privacy (EuroS&P), 216–235. https://doi.org/10.1109/EuroSP48549.2020.00022

Lipmaa, H., & Toft, T. (2013). Secure Equality and Greater-Than Tests with Sublinear Online Complexity. In F. V. Fomin, R. Freivalds, M. Kwiatkowska, & D. Peleg (Eds.), Automata, Languages, and Programming (pp. 645–656). Springer Berlin Heidelberg.

Goldreich, O. (2001). Foundations of Cryptography: Volume 1, Basic Tools. Cambridge University Press. https://books.google.co.uk/books?id=H1x7MgEACAAJ

Goldwasser, S., Micali, S., & Rackoff, C. (1985). The Knowledge Complexity of Interactive Proof-Systems. Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, 291–304. https://doi.org/10.1145/22145.22178

Bitansky, N., Canetti, R., Chiesa, A., & Tromer, E. (2012). From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again. Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, 326–349. https://doi.org/10.1145/2090236.2090263

Parno, B., Gentry, C., Howell, J., & Raykova, M. (2013). Pinocchio: Nearly Practical Verifiable Computation. https://eprint.iacr.org/2013/279

Teor. (2021). Zebra: Zcash Zero-Knowledge Proofs at Scale. https://zkproof.org/2021/06/03/zebra-zcash-zero-knowledge-proofs-at-scale/

Setty, S. (2019). Spartan: Efficient and general-purpose zkSNARKs without trusted setup. https://eprint.iacr.org/2019/550

Ben-Sasson, E., Bentov, I., Horesh, Y., & Riabzev, M. (2018). Scalable, transparent, and post-quantum secure computational integrity. https://eprint.iacr.org/2018/046

Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., & Maxwell, G. (2017). Bulletproofs: Short Proofs for Confidential Transactions and More. https://eprint.iacr.org/2017/1066

Labs, M. (2023). Repo: Awesome Zero-Knowledge Proofs. https://github.com/matter-labs/awesome-zero-knowledge-proofs

Finogina, T., & Herranz, J. (2023). On remote electronic voting with both coercion resistance and cast-as-intended verifiability. Journal of Information Security and Applications, 76, 103554. https://doi.org/10.1016/j.jisa.2023.103554

Fouque, P.-A., Poupard, G., & Stern, J. (2000). Sharing Decryption in the Context of Voting or Lotteries. Proceedings of the 4th International Conference on Financial Cryptography, 90–104.

Boudot, F. (2000). Efficient proofs that a committed number lies in an interval. Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques, 431–444.

Zama. (2022). TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data. https://github.com/zama-ai/tfhe-rs

Appendix A – Glossary

Political Terminology

Constituent: An individual within the system with a right to a single delegable vote

Delegate: An individual elected to act on behalf of another in their absence

Liquid Democracy: A system of democracy representing a blend between representational and direct democracy via a continuously revocable delegable vote

Vote/Referendum: A way of making a decision by asking a group of people to cast ballots

Decentralised Technology

Decentralised: A system in which there is no single authority with executive power

Digital Anonymous Organisation (DAO): An alternative legal/organisational structure for communities with no central governing body. Used to make decisions following a 'bottom-up' approach

Peer to Peer Network: A networking approach that relies on equally powerful nodes bidirectionally communicating to perform network tasks, in contrast to traditional client-server architectures

Smart Contracts: A self-executing program that automates the actions required in an agreement or contract, used to automatically perform actions on a blockchain without user control

Knowledge Proofs

Prover: The party attempting to convince a verifier of knowledge

Verifier: The party being convinced of the truth of a statement

Zero-Knowledge (zk): A property of a knowledge proof meaning that no additional private information is exposed as part of the proof

Witness: A piece of information which can efficiently verify that a statement is true. In general this is the secret information which a zk protocol is trying to hide, such as a private key.

Committment: A sealed, unmodifiable committent to a given value, used to enforce honest reporting of ciphertexts

Other

Differential: A representation of the changes between one version of a document and the next