Länsstyrelsen Skåne

Interreg North Sea Region BLING
European Regional Development Fund

Blockchain-based e-voting system without digital ID: A Proof-of-Concept

Blockkedjebaserat elektroniskt röstningssystem utan digitalt ID: En konceptvalidering

Leonard Schick

Fakulteten för hälsa, natur-och teknikvetenskap

Datavetenskap

C-uppsats 15hp

Supervisor: Leonardo Martucci

Examiner: x

Date: 2023-10-01

Abstract

Electronic voting systems have the potential to offer a cost effective, secure and transparent way of communicating with the citizens, increasing trust and participation.
However creating a secure open source electronic voting system providing confidentiality and transparency with sufficient performance has long been a challenge.
This thesis proposes a Proof-of-Concept (PoC) for a blockchain-based e-voting system in the absence of government-approved digital ID, aiming to provide a resource for public actors, offering a functional smart contract implementation and suggests an infrastructure design it can utilize. The infrastructure design for the PoC features Hyperledger Besu in a permissioned configuration using PoA (QBFT) algorithm with 14 nodes. The voting process involves: account generation and distribution via mail by the government, voter-created passwords encrypting browser generated wallets with voting rights acquired by blind signatures. These components work in conjunction with a smart contract, which serves as the central mechanism for handling the voting process. The thesis finds the system meets the key criterias for an evidence-based e-voting system to a high degree but require testing of the infrastructure design together with the smart contract to assess the performance in order to determine the practical feasibility.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Election results have historically been questioned due to claims of fraud and manipulation, leading to widespread distrust. For example, there were claims of vote buying and irregular vote counting in the 1824 US Presidential Election. The controversial Florida result in the 2000 Presidential Election raised concerns about the validity of the vote count.[1] More recently, due to newly introduced mail-in ballots and electronic voting machines using closed source software, the 2020 Presidential Election was met with a substantial level of suspicion and scrutiny of the election process and its outcomes.[2]

The use of a blockchain-based polling system carries potential advantages. A key benefit is the provision of a more transparent and secure way to conduct polls. Malicious actors may find it more challenging to interfere with the polling process or results due to the nature of blockchain as a decentralized, distributed ledger, capable of providing a verifiable audit trail. These properties can increase public confidence in the polling process and the credibility of the outcomes.

The ability to use a mobile app to vote from anywhere, could also increase voter turnout. It could also reduce the environmental and economic costs of the election due to less resources needed, such as paper ballots, fuel for transportation, or use of venues.

This thesis is of interest because blockchain can be used to develop the dialogue between the citizen and society by enabling an effective and convenient way of opinion gathering. A working blockchain solution for elections and polls would be of great interest and value to actors in the public sector. Additionally, blockchain technology has the potential to add transparency and integrity to the voting process.

1.1 Problem Statement

One of the greater challenges when developing a blockchain application is choosing the infrastructures that support the blockchain application, i.e the blockchain itself, that is suitable for e-voting. This is of significant importance, as the configuration of the blockchain parameters affect the practical legitimacy of the e-voting application, for which the reasoning is the following.

A practical e-voting solution must be scalable enough to handle the throughput requirements of the use case, i.e the volume and speed at which voters cast their ballots. It must also have a high level of security to ensure the integrity of the votes and inspire faith in the voting process. Additionally, it should be somewhat decentralized, or else there would be no significant added benefit to using a blockchain over a centralized database.

Blockchains are generally constrained to being especially robust in two out of the following attributes, at the cost of the third: [link to blockchain trilemma section]

- decentralization
- security
- scalability.

With these compromises in consideration, one needs to carefully design the network in such a way that it satisfies the demands of an e-voting application, both with regards

to network performance and security, but also considering the systemic risks when accounting for internal malicious actors.

Besides considering the technical aspects of the network, there is also the added precondition that it should be open source, otherwise it's not auditable to the public and possibly even the government, in the case where it's proprietary code from a private company. Open source is the most transparent and available option for governments at this stage, since there also is a low supply of blockchain based e-voting solutions in general, proprietary or otherwise. This leaves most governments with no options, since using proprietary code from another country is highly questionable due to the heightened potential risks of foreign intervention, lack of internal oversight and judicial authority.

Beyond the network itself, there is the application responsible for the logic of the voting process. The development of a smart contract that meets the specific requirements of e-voting presents its own set of challenges. To satisfy the specifications of current traditional voting systems, it would need to guarantee the following functionalities: <insert spec from google docs here>

Since the implementation of the smart contract largely depends on what environment it would be executed in, i.e the chosen blockchain, a general design is not sufficient as a PoC, as the design might not be implementable in the chosen execution environment. Thus a minimal working example would be needed for it to suffice as a PoC.

This thesis proposes and evaluates the use of blockchain as a service for e-voting by implementing a voting application in the form of a smart contract.
The thesis makes the following original contribution:
proposes a PoC for a blockchain-based e-voting system that doesn't use an ID authentication application

1.2 Aims and Objectives

The goal of this thesis is to be a useful resource for public actors who wish to develop their own e-voting system using blockchain technology. The objective is to provide a proof of concept in the form of a smart contract and network design that can be worked from and use in a full stack implementation of the application, i.e integrated into the network with a front-end.

The deliverable of this project is the proof of concept e-voting blockchain application, which will include a fully functional smart contract written in Solidity, as well as the design for the infrastructure required to execute it. The success of the project will be measured by the degree to which the application and it's blockchain design satisfy the commonly used minimum requirements for traditional voting.

1.3 Ethics and Society

There are a couple of social and ethical issues that have been raised with regards to the development and implementation of e-voting systems. The commonly mentioned issue is what is known as the "digital divide". The digital divide refers to the idea that access and participation to computers and the internet, is unevenly distributed between the different demographics in society, because factors such as age and socioeconomic status can influence technological literacy. From metastudies of the current state of research regarding internet voting, this is mostly a problem of the past due to the increased adoption of smart phones and internet access, as well as a shrinking population who are old with a strong association with technological illiteracy. Viewed from another perspective, being able to vote online from a smartphone or computer has the potential to increase the accessability and participation due to the low cost of time and commitment to vote, as there is no travel cost, time or planning.

The trust is also an issue concerning e-voting. If there are doubts - based or un-

founded - about the integrity and accuracy of the voting system, it could undermine the trust in the democratic process. Traditional systems using paper ballots are easy to understand and verify for everyone, to the contrary only people with technological expertise would be able to, with online voting using open source software. This only exacerbates potential problem previously mentioned about how the democratic process may be perceived as dubious. Although the problem of trust seem to be insuperable, it's worth mentioning that some currently used voting systems incorporate e-voting machines, such as in the USA and Canada, that use close source software, which brings doubts and concern even to the experts researching the field of electronic voting security.

1.4 Methods

For a new e-voting system to be used, it needs to at least approximate the standards of currently used voting systems. The first step in the project was to define the requirements for voting and how that translates into e-voting. There is a set of minimal election security requirements that voting systems should fulfill in order to meet the standards of the current voting systems in countries that are deemed to have intact evidence-based voting systems.

The standard usually include, but are not limited to the following requirements:[]

- *Confidentiality:* The secret ballot, i.e vote is cast anonymously
- *Auditability:* Ability to verify that votes were collected, counted and recorded as intended.
- *Contestability:* Ability to provide publicly verifiable evidence of when an error is detected.
- *Decentralization:* The degree to which the control of the system is distributed to different stakeholders.

In addition to these properties, there is also a need for a sufficient performance of the system for it to be practical. The network capacity needs to be scalable enough to handle the throughput demand for the use case, as well as be cost effective relative to currently used systems. Although this paper didn't test the performance, it was taken into consideration throughout the design and implementation.

After having established the set of evaluation criterias for a system which is suitable for e-voting, a search for an open source blockchain which meets the criterias to the highest degree began. This search was conducted by reviewing the literature regarding blockchain voting applications and noting which platforms are commonly used. Then searching for open source platforms found through google and GitHub searches using terms such as"open source blockchains", "open source blockchain projects", "open source distributed ledger", etc to see if there were viable alternatives.

When the blockchain which was most suited for the application had been chosen, the development of said application began. The development was conducted with Remix, an online IDE with a Solidity compiler. All functionalities were unit tested during and after the development, again using Remix IDE.

After the smart contract was fully functional, it was unit tested and deployed on a testnet, an instance of the chosen blockchain, which acts as a test environment. The deployed smart contract was integration tested by simulating a small scale poll with 10 voters, to ensure fault-free functionality in an environment similar to the intended execution environment.

1.5 Stakeholders

The stakeholders for this degree project include the County Administrative Board of Skåne, which is part of the partnership involved in the implementation of the project "Blockchain in Government" (BLING). The project is funded by the EU program In-

terreg North Sea Region and aims to explore the potential of blockchain technology in the public sector. The project's lead partner is the city of Groningen in the Netherlands, and it involves 12 other parties in the North Sea region, including universities such as Edinburgh, Aalborg, Gothenburg, and Oldenburg.

The County Administrative Board of Skåne has a stake in this degree initiative because they believe blockchain technology has the ability to enhance the public serviceăsector's offerings. They are interested by the idea of using a blockchain-based polling system because they think it may improve the dialogue between people and society and be useful to public figures.

The idea behind the initiative is that using blockchain technology can increase the voting process's transparency, integrity and availability, which will ultimately boost people's confidence and faith in the democratic system. As such, they view this degree project as an opportunity to delve deeper into blockchain technology and explore its potential to improve public service delivery and increase the efficiency and effectiveness of government operations.

This degree project will be of benefit by providing a PoC smart contract and network design that they can work from and use in a full stack implementation of an e-voting application, i.e integrated into the network with a front-end.

1.6 Delimitations

The thesis does not cover the full implementation and setup of the blockchain, but rather attempts to provide a description of the basic parameters required to establish a secure e-voting system using the Hyperledger Besu blockchain as the example architecture. The description of the blockchain will be one of a high-level configuration of the network, which will include the validator node setup, if the network should be permissionless or permissioned, public or private, and other parameters such as gas fees.

The smart contract functionality will be tested in a limited environment that doesn't reflect the same level of demand expected to be seen when used in a real context. This is mainly the case since simulating a 1:1 of the network configuration that would be used live, is outside the scope of the project. Even though this would make for a more robust PoC, the small-scale testing done in a general network configuration such as an Ethereum testnet is sufficient for a PoC, as the execution engine is the same, regardless of the network configuration.

As the network configuration affects the performance and security, and the testing is not done with the intended configuration outlined in the project, this leaves the network performance and security testing outside the scope of the project.

In addition to the aforementioned limitations, this thesis also acknowledges certain constraints in relation to the stakeholders and the specific requirements they have put forth. As the stakeholders are located in Sweden, a country without a government authentication app, an alternative means of authentication must be used to verify and distribute voting rights until such an app is developed. In the interim, stakeholders have indicated their ability to provide individuals with voting rights the necessary details for participating in the voting process, such as login or public- and private keys.

Furthermore, the stakeholders have expressed their desire for the nodes to be hosted on national servers. This requirement aims to minimize the risk of foreign interference while ensuring complete judicial control and responsibility over all entities involved in the system. Consequently, this constraint must be taken into account when designing the e-voting system.

1.7 Disposition

The following chapter will give a brief introduction to Blockchain and make a review of prior research and literature regarding e-voting blockchain applications. Chapter 3

goes through the design process, while chapter 4 contains the implementation of said design. Thereafter the result is showcased in chapter 5 and lastly ending the thesis with the conclusions in chapter 6.

Chapter 2

Background

A blockchain is a distributed ledger technology that records transactions and data in a network of computers. The word “blockchain” refers to the data structure, that consists of blocks that essentially are a collection of transactions details, such as timestamps and cryptographical signatures of senders and receivers. Each block is linked to the previous one with a cryptographical signature, which creates a tamper-proof and easily verifiable public record containing all transactions, that is stored as a chain, hence the name blockchain.[]

Blockchain has many potential use cases such as for example supply chain management, digital ID, intellectual property rights, voting systems and so on but are commonly associated with cryptocurrencies, that is likely a result of its history that was initialized by its potential use case as a form of decentralized electronic cash, as will be showcased in the following section.

History

The invention of the blockchain was the culmination of decades of work in the field of cryptography. Beginning in 1976, the idea of public key cryptography was introduced by Martin Hellman, Ralph Merkle and Whitfield Diffie at Standford University.[3] The

same year a paper called ”New Directions in Cryptography” brought up the notion of a distributed ledger, a way to distribute and store information in a decentralized way. Later that decade Merkle trees where invented, a data structure which allows for an efficient and secure way of cryptographically verifying the contents of a large data structure. In the early 1980's, David Chaum made significant contributions with papers such as “Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms”, “Computer Systems Established, Maintained and Trusted by Mutually Suspicious Groups” and “Untraceable Electronic Cash”, introducing the idea of anonymous communication networks and anonymous electronic money.[4]

Some additional key concepts were added in the 1990's, “How to time-stamp a digital document” by Stuart Haber & W. Scott Stornetta, which demonstrated how tamper-proof timestamping could be achieved, while also later implementing Merkle trees in their design, joining two concepts used in most blockchains today.

Later on Adam back implemented another key concept he called“Hashcas”, which was based on the idea“... to require a user to compute a moderately hard, but no intractable function..” introduced in a paper in 1992 called“Pricing via Processing or Combating Junk Mail”. Hashcash was initially a solution to spam email by attaching stamps to emails which required a computational cost to create, today the algorithm for the stamp creation is more known as a proof-of-work algorithm (PoW), which is used in many of the biggest blockchain projects, such as Bitcoin and up until 2022, Ethereum as well.[5][6]

Hashcash was soon after it's creation used by Wei Dai to create “b-money”, which was a anonymous electronic cash system based on a peer to peer network. Although the project aimed to provide what toady's cryptocurrencies offer, it never officially launched.[7]

The Bitcoin whitepaper was published in 2008, ten years after b-money's creation, written by the pseudonymous person or group of people known as Satoshi Nakamoto.

The paper presented the concept of a decentralized, peer-to-peer electronic cash system, called Bitcoin, with the focus on the ability to transfer and store value over the internet without a trusted third party. The bitcoin whitepaper proposed a novel implementation of the previously mentioned cryptographical techniques and data structures that solves the double-spending problem using a peer-to-peer network, to create what is later to be referred to as a blockchain.

Bitcoin uses a distributed ledger in a peer-to-peer network to record transactions and prevent double-spending. It introduced the concept of "mining," which is the process by which new bitcoins are created and transactions are validated by a network of nodes in the system, using the PoW. The incentive to validate transactions correctly is regulated by the PoW algorithm which validator nodes need to run, i.e earning new bitcoins when validating transactions correctly and losing money if deviating from the consensus of the majority, as the PoW inflicts a computational cost, which translates into used electricity.

There are 15 years later roughly 22800 blockchain projects are registered on coinmarketcap - one of the leading blockchain token tracking websites - and a total market capitalization of 950 billion dollars. Although most of them are assumed to be copies of already existing projects and have no real ambition to develop real applications, it's a testament to the pace of growth that the blockchain industry has seen since Bitcoin's creation.[8] The following section is intended to be an introductory overview of how blockchain function.

2.1 Blockchain: The Basic Structure and Network

Fundamentally, blockchains can be described in two parts, their networks and their data structures. This section will first give a general overview of the data structure and then go on to the different configurations of the network and how it affects its behaviour.

A blockchain is a continuously expandable linked list of data records in individual

blocks, shared in a peer-to-peer network. New blocks are created after a consensus procedure and attached to an existing chain using cryptographic methods. Each block typically contains a cryptographically secure hash (digest) of the previous block, a timestamp, and transaction data.[9]

A blockchain can be used in accounting when consensus on the current and error-free state must be established in a decentralized network with many participants. What is documented is irrelevant for the concept of the blockchain. The crucial point is that later transactions build on previous transactions and confirm them as correct by proving knowledge of the previous transactions. This makes it impossible to manipulate or delete the existence or content of previous transactions without also changing all subsequent transactions. Other participants in the decentralized accounting recognize a manipulation of the blockchain by the inconsistency of the blocks.[10]

It must be ensured that an identical chain is created for all participants. For this, proposals for new blocks must first be developed. This is done by the validators in the network, who then must agree on which proposed block is actually inserted into the chain. This is done through a so-called consensus mechanism, also sometimes called a consensus protocol, an algorithmic procedure for voting. [11]

This report will briefly explain some of the more commonly used ones, namely proof of work, proof of stake and proof of authority, but it's worth mentioning that there are many different variations of these, usually modified and tweaked in ways that often benefit the performance of the blockchain with regards to its application.

PoW

The Proof-of-Work consensus mechanism is based on cryptographic proofs that require the solution to a complex computational problem to generate a new block on the blockchain, however verifying the solution of the computation is simple and requires minimal resources. The mechanism's fundamental purpose is to limit the number of

blocks produced, by demanding that nodes perform computationally intensive work, usually involving a hash function, before appending a block to the chain.

Upon discovery of a solution, a node can broadcast it to the network, and other nodes can verify it instantly. The computational work required to find the solution ensures that appending a block is a challenging and resource-intensive task, thus hindering network spamming or overload with excessive blocks.[11]

PoS

In PoS, the validator's task is similar to that of miners in Proof-of-Work (PoW). However, rather than solving intricate mathematical problems, validators must stake their tokens to take part in the network. This makes PoS more energy-efficient in contrast to PoW, where significant computational power is required to create new blocks.

The PoS algorithm picks validators at random to validate transactions and create new blocks. The probability of being chosen is proportional with the amount of tokens staked by the validator, i.e the more tokens staked, the higher the probability of being chosen. After being selected, the validator submits the next block and broadcasts it to the network. Other validators then authenticate the block and add it to the blockchain. Validators who successfully add blocks are compensated with additional tokens.

PoS encourages validators to act in the best interest of the network by having penalties for misconduct. Validators who fail to adhere to the regulations or behave maliciously may have their staked tokens reduced as a penalty. This mechanism promotes healthy network behavior and helps shield the network from attacks.[12]

PoA

Unlike the PoW and PoS mechanisms, the PoA consensus algorithm leverages the value of identities, meaning that block validators do not stake their coins or spend high amounts of computational power but instead stake their own reputations.

Meaning blocks are verified by pre-approved participants who function as moderators for the system. Thus, the security of PoA blockchains is ensured by the validating nodes that are arbitrarily chosen as trusted entities, therefore reducing the computational costs of arriving at consensus, as there is inherent trust in the validators from the beginning by the very fact that they were chosen.

In the PoA model, a limited number of block validators are employed, this combined with the fact that the chosen validator nodes are inherently trusted to some degree, makes it an effective system with good performance. The PoA consensus mechanism sacrifices decentralization in favor of achieving high throughput and security.[12]

Public vs private networks

The general definition in the context of blockchain networks is that in public networks allow anyone to participate, meaning anyone can view the contents of the blockchain and submit transactions. In contrast, private networks have an invite-only policy, where only a selected group of participants can access the network.[12]

However, in the context of ethereum networks, any network which is not connected to the ethereum mainnet or a testnet, is considered a private network.[13] To maintain consistency in terminology throughout this thesis, we will use the general definition of public and private blockchain networks as described earlier.

Permissionless vs permissioned networks

Permissionless networks refer to networks where any member of the network can participate in the consensus process, i.e validate and add blocks to the chain, while permissioned networks only allow those who are “permissioned”, i.e arbitrarily allowed the privilege to write to the chain by validating transactions. This privilege is usually distributed with certificates that are then tied to the network identities, usually managed by a certificate authority.

These network attributes can be combined into different configurations, such as public permissioned networks, private permissioned networks and so on, giving rise to a multitude of different networks, suitable for different use cases.

Blind Signatures

Blind signatures are an important concept in cryptography, helping to maintain anonymity in digital transactions. They are commonly used in systems like electronic voting and anonymous credentials.

The main idea behind blind signatures is that a message can be signed by a party without revealing the message's content to the signer. This is typically done in three steps:

i. **Blinding**: The sender changes their original message into a different form, making it unrecognizable. This ensures the signer can't see the actual message.

ii. **Signing**: The signer, without knowing the message's content, signs this changed version.

iii. **Unblinding**: After receiving the signature, the sender changes the message back to its original form, now with a valid signature.

Throughout this, the signer doesn't know the content of the message, ensuring the sender's message remains anonymous.

2.1.1 Ethereum and Smart Contracts

Ethereum is a decentralized, permissionless network based on the blockchain technology of Bitcoin. In addition to the ability of transferring value in the form of its currency (similar to Bitcoin), Ethereum also allows for the execution of smart contracts, which

are essentially programs stored on the blockchain. These allow for the creation of so-called Decentralized Applications (Dapps).[13]

Smart Contracts can hold a balance and send transactions over the network, but they are not controlled by a user. Instead, they are provided by the network and executed as programmed. User accounts can then interact with a Smart Contract by submitting transactions that trigger a function defined in the Smart Contract. Smart Contracts, like traditional contracts, can define rules and automatically enforce them through programming. By default, Smart Contracts cannot be deleted and interactions with them are irreversible.[14]

The Ethereum Virtual Machine (EVM) is the runtime environment that executes Smart Contracts on the Ethereum network. It is a sandboxed environment that ensures code execution is secure and deterministic. The EVM defines a set of instructions that Smart Contracts can use to perform operations, and it is responsible for calculating the new state of the blockchain after each Smart Contract execution. This new state is then broadcasted to all nodes on the network - similar to a transaction - to maintain consensus.

The state machine's continuous, uninterrupted, and immutable operation is the sole goal of the Ethereum protocol. All Ethereum accounts and Smart Contracts exist in this environment. Ethereum has exactly one valid state for each block in the chain, and the EVM determines the rules for calculating a new valid state from block to block.[13]

The following section will review some previous research done about blockchain e-voting applications.

2.2 Blockchain-based e-voting systems

Hjálmarsson and Hreiðarsson [15] conducted a comprehensive study on the potential of distributed ledger technologies, particularly blockchain, for implementing electronic

voting systems. They evaluated several popular blockchain frameworks and proposed a novel electronic voting system based on a permissioned blockchain that enables liquid democracy. Among the blockchain frameworks they assessed, including Exonum, Quorum, and Geth, they ultimately chose Geth as the foundation for their work, one of the first implementations of ethereum.
This decision was based on Geth's developer-friendly nature and its ability to execute smart contract exactly as programmed without the possibility of downtime, censorship, fraud, or third-party interference. Geth can be implemented as a public or private network and since it uses the EVM, it supports all smart contracts written in solidity, making it a versatile choice for electronic voting systems.

Their proposed e-voting system is configured as a permissioned network that utilises the PoA consensus algorithm called *Clique*, and the validator nodes are run by election administrators, either government agencies or institutions. The election administrators create the election smart contract which sets the candidates, length of the voting period and the districts.
The voting process implements secure authentication through an Icelandic service provider called Auðkenni, which uses proprietary software and RFID scanners for identity verification. Voters authenticate themselves by scanning their electronic ID and entering a corresponding PIN number.
Once a voter is authenticated, they interact with a smart contract for the ongoing election, which lists the candidates they can choose from. After selecting a candidate and casting their vote, the voter signs their vote by re-entering their electronic ID's PIN number. The vote data is then verified by the corresponding district node, and if accepted, it's broadcast to the rest of the network to be agreed upon by the majority of the district nodes. When consensus is reached, the voter receives a transaction ID in the form of a QR code, which can also be printed. The smart contract adds one vote to

the chosen party, and this functionality is used to determine the election result in each voting district.
Hjálmarsson and Hreiðarsson's design is particularly advantageous because it is built on a blockchain infrastructure that utilizes the EVM, can be configured to be public or private and allow for multiple consensus algorithms. This choice of infrastructure ensures that their e-voting system benefits from the extensive support provided by the robust Ethereum-based blockchain developer community. As a result, the system can take advantage of the great body of existing knowledge, tools, and resources, enabling faster development, easier maintenance, and enhanced security. Additionally, the thriving community fosters continuous improvement and innovation, which can help the e-voting system to stay relevant and adapt to new challenges in the future.

In the study by McCorry, Shahandashti, and Hao, the authors presented a small scale e-voting application, implemented as a Solidity smart contract on the Ethereum blockchain [16]. Their implementation was tested on an Ethereum test network with forty simulated voters, demonstrating the potential for minimal setup elections at a reasonable cost of $0.73 per voter.

The implementation assumes that voters and the election administrator have their own Ethereum accounts, with the Web3 framework facilitating communication between a user's web browser and their Ethereum client. The election administrator, represented by a designated owner, was responsible for authenticating voters and updating the list of eligible voters. The smart contract enforced a series of timers to guarantee timely election progress, allowed only eligible voters to register and vote, and could require voters to deposit ether upon registration, refunding it upon vote acceptance.

They suggest future work to explore the feasibility of running a national-scale election over the Blockchain. Its said that the implementation may require a dedicated Blockchain, such as an Ethereum-like blockchain that exclusively stores the e-voting smart contract. This new blockchain could have a larger block size to store more

transactions on-chain and could be maintained in a more centralized manner.

The voting application was implemented as a Solidity smart contract with a web page as the front-end for users, making it modular and adaptable. Meaning the application can be executed on any EVM-compatible blockchain, allowing for broader adaptability. The use of Ethereum mainnet as the chosen infrastructure to run the application is not an effective solution due to the high cost and low throughput on the Ethereum mainnet. The cost of $0.73 per voter was achieved in 2016, when the cost of execution on the mainnet was significantly lower compared with today, making the choice obsolete. The assumption that the voters have their own Ethereum accounts and web3 frameworks is not a realistic one in a real use case for the general public. These functionalities should have to be integrated into the application for it to align with the context of the use case, as technical knowledge cant be a prerequisite for voting.

Tanröver and Ta conducted a systematic review of blockchain-based electronic voting systems, aiming to provide an overview of the state of the research in the field, identify associated challenges, and forecast future directions [17]. They reviewed 63 research papers that advised the adoption of blockchain frameworks for voting systems and highlighted that blockchain-supported voting systems could offer different solutions compared to traditional e-voting systems.

Regarding the blockchain platforms and consensus models used in e-voting systems, Tanröver and Ta found that most studies focused on the general idea of blockchain-based e-voting without providing explicit technical details and implementation proposals. Among the analyzed papers, Ethereum clients represented the most preferred platform (24%) followed by "Hyperledger", according to the authors, it may be due to its smart contract capabilities, flexibility, wide adoption and all the benefits that comes from it. They mention the Hyperledger project without specifying which of the platforms within the project.

In their discussion of future research directions for e-voting with blockchain plat-

forms, they emphasized the need to resolve concerns about possible violations of election rules specified in smart contracts or election results. One possible solution they suggested was using private blockchains, although ensuring transparency in such cases remains a challenge. Additionally, they highlighted scalability as a major concern, particularly when the system performance decreases with a high rate of execution. As an example, they mentioned the 2018 elections in Turkey, where the election authority declared 59,369,960 voters, necessitating a transaction rate of at least 2061.46 votes per second according to rough calculations, for a functional blockchain-based e-voting system.

The study by Tanröver and Ta highlights Ethereum-based blockchains as the majority choice for e-voting applications. It is important to note that Ethereum was the first blockchain enabling smart contract functionality, making one question if there is bias to the data due to the adoption lag of new platforms, as many more blockchains have been created in recent years that build on the EVM but has other optimizations than that of Ethereum itself, which could be advantageous in certain contexts regarding e-voting applications. The suggestion to use the private blockchains to minimize risks of attacks is a sensible one as it could offer an effective solution to some risks by isolating privilege to trusted nodes. Regarding the transparency concern as a result of using a private network, one potential solution could be to use a blockchain which allows one to implement a permissioned network configured with public read permissions but invite-only write permissions. This setup would allow the general public to verify the integrity of the voting data while restricting the ability to submit transactions as unauthorized participants.

In a paper by Caixiang Fan et al. they analyze the performance of Hyperledger Besu, focusing on the impact of different factors on transaction throughput and latency[18]. The paper emphasizes that workload type is a crucial factor influencing the Besu network performance. Three different types of workloads are defined: (i) open transac-

tions: one write operation to a mapping in a smart contract (ii) transfer: a transfer of ether from one account to another (iii) query: performs a single read operation

Its stated that open transactions consistently demonstrate better performance than transfers, thought to be due to the fact that open transactions require one write operation while transfers require two. Consequently, open transactions consistently outperform transfer transactions. With default configurations, Hyperledger Besu can achieve a baseline transaction throughput of approximately 1500 req/s for open transactions only, and 1000 req/s when both open transactions and transfers are included.

The scalability analysis reveals that Besu has limited vertical scalability in terms of node size, and it does not horizontally scale well with an increasing number of validators. Furthermore, the study highlights the importance of configuring block time according to an application's needs, as it directly affects the latency and throughput of Besu.

They also observe that the three PoA consensus algorithms - Clique, IBFT 2.0, and QBFT - have negligible effects on transaction throughput compared to other operations, such as transaction execution and blockchain state updates, in a small private blockchain network with 8 nodes, with open transactions demonstrating superior performance in terms of throughput and latency across all three algorithms. Although this is the case for a network with 8 nodes, QBFT was observed to have slightly better scalability than the other PoA algorithms, notably up to 14 nodes without obvious performance loss. They conclude that node computation resources have a limited impact on consensus time in a Besu network with a fixed number of validators (e.g., 8).

The findings of Caixiang Fan et al. indicate that the throughput and scaling of a private Hyperledger Besu network could be sufficient for an e-voting application in smaller countries. By doing the same rough calculations as Tanröver and Ta did with Turkeys national election but with Sweden's in 2020, 7 775 390 had voting rights, of which 84% voted, i.e 6531327 votes [19]. If everyone were to cast their vote over an

8 hour period, it would still only be an average of 6531327/60/60/8 227 req/s. The required throughput of the network would in reality be much lower as not everyone vote on the same day, but this gives an estimate or benchmark, of an acceptable throughput for smaller countries.

Regarding the observation of performance effects on workload type, like the authors note, the performance has the strongest negative correlation with the number of write operations per transaction. Which would also be expected, as write operations or state changing operations, are usually more complex and require more computational resources than read operations or pure compute. Storage operations require all nodes to update or expand their state database [20]. The fact that they state that Besu has a limited vertical scalability in terms of node size, might sound like a pure negative, although it can be positive for potential stakeholders as it in a way lowers the barrier to entry by achieving maximum performance at a lower cost of hardware. The downside is if that maximum performance is insufficient for the stakeholders use case.

Chapter 3

Design

3.1 The Voting Process

Since one of the limitations of this design is the absence of a government-issued digital ID application for ID authentication and distribution of voting rights, an alternative solution has been created.

(i) *Authentication:* The government generates unique account details for their website that serves as the interface to the blockchain application. These account details consist of randomly generated unique strings of a specific length. The government then randomly distributes these account details to eligible citizens - i.e those with voting rights - by sending them via mail.

(ii) *Distribution of voting rights:* Upon their first login to the website, voters are prompted to create a secret password to be used when casting their vote. After setting up the password, an anonymous wallet is generated within the voter's browser, given voting rights by the website via a blind signature scheme to hide the voters address. The voter's browser subsequently whitelists the address in the voting contract using said blind signature. This wallet is then encrypted using the

voter's secret password before being stored on the government servers.

(iii) *Vote:* When a vote is cast, the corresponding address is marked as "Voted" in that poll contract in order to guarantee one vote per person per poll.

The voting process has been designed with the primary goals of ensuring security, transparency, and anonymity while adhering to the limitations and constraints outlined in the methods and limitations sections of the thesis. The rationale behind the specific design choices is as follows:

(i) *Generating and distributing unique account details:* This method of distributing voting rights involves creating random account details, ensuring that no individual's identity can be inferred from the account details alone. Malicious actors within the government would have to intervene in the account detail distribution process to map the identities to the randomly assigned account details. Still this would not be enough to tie an identity to a vote, as the voters address is not tied to the account details. Sending these account details via mail to eligible citizens offers a secure and reliable distribution channel while working within the constraints of not having a government-issued digital ID application.

(ii) *Wallet creation and management:* Prompting voters to create a secret password adds an extra layer of security, decreasing the risk of anyone but the true account owner being able to use their vote. Encrypting the wallet using the voter's secret password before storing it on the government servers further enhances security but more importantly keeps the government issued account from being associated with the voters address, which will be publicly visible on the blockchain once they cast their vote; therefore maintaining the voters anonymity. The wallet generated in the voter's browser and whitelisted in a smart contract provides anonymity, as the wallet creation is created locally and is given voting rights without the server

ever seeing the address, it's not directly tied to the voter's government account or personal information.

(iii) *Marking addresses as "Voted":* This mechanism ensures that each voter with voting rights is only able to cast one vote per poll, maintaining the integrity of the voting process.

3.2 The Blockchain

From the research presented in the background section and accounting for the time available for the development of the application, it was concluded that it would be beneficial to use an EVM-compatible blockchain due to their large support and the EVM's long track record. Therefore the smart contract will be written in the Solidity language.

There are many different Ethereum clients to choose from, since they all use the EVM to execute the smart contracts and adhere to the Ethereum protocol, the differences usually lie in the features and configuration options, which also can impact the performance. Meaning that the choice of Ethereum client used is of less importance as long as it meets the demands put on by the use case. As noted in the literature review Ethereum-based platforms were the most used for voting systems and the Hyperledger project was the second most used from the 63 papers they analyzed, and Hyperledger's only Ethereum client is Hyperledger Besu.

Hyperledger Besu is an open-source Ethereum client developed under the Apache 2.0 license and written in Java. It supports the EVM, meaning that it can execute the same smart contracts and transactions as the Ethereum mainnet. Besu can be configured as a public or private network and supports the following consensus algorithms: PoW,

PoS and PoA (IBFT 2.0, QBFT, and Clique). It offers features such as privacy and permissioning through integration with the Hyperledger Besu Network Manager, which simplifies the deployment and management of permissioned networks.

Hyperledger Besu was therefore chosen as the example blockchain, as it meets the criterias of the context of being an Ethereum-client that is open-source, highly configurable, has data on performance metrics that indicate a sufficient scalability for the stakeholder's use case, written in a widely known language and maintained in a collaborative effort hosted by the Linux foundation.

3.2.1 Network configuration

Note: As mentioned in the background chapter, to maintain consistency, we will use the term "public" to refer to a blockchain network that is open to anyone to participate in, rather than the definition in the context of Ethereum networks, where it would be called a consortium or private network.

The recommended configuration for this design was chosen to be a public permissioned blockchain. This choice was made to strike a balance between transparency, security, and control while adhering to the constraints outlined in the methods and limitations sections of the thesis.

1. *Public:* The public nature of the blockchain ensures transparency by allowing the general public to independently read and verify the vote data on the blockchain. In contrast, a private network would require the public to rely on an interface to a node provided by the government, potentially raising trust issues.

2. *Permissioned:* The permissioned aspect of the network is primarily a result of the limitation that nodes must be run on national servers. However, this configuration also provides several potential benefits:

(i) *Enhanced reliability and performance:* A more reliable network with better performance, as the validator node setup is static, which allows for efficient communication and resource allocation among the validators

(ii) *Improved security:* Manually chosen validators by trusted authorities mitigate risks of external manipulation, such as Sybil attacks, ensuring a higher level of security and trust in the network.

(iii) *Streamlined governance and updates:* Permissioned networks enable a more efficient decision-making process among chosen validators, allowing for timely upgrades and maintenance without the risk of hard forks or other disruptive events that may occur in permissionless networks.

(iv) *Regulatory compliance:* Increased compliance with legal and regulatory requirements, as the permissioned network can be more easily adapted to meet specific requirements for data privacy, security, and transparency, allowing for a more tailored approach to address the needs and concerns of stakeholders in the voting process;

(v) *Reduced susceptibility to attacks:* Permissioned networks can implement ratelimiting and other security measures more effectively, reducing the risk of spam and DDoS attacks and ensuring the integrity and availability of the voting system for its intended purpose.

A permissioned network reduces the decentralization and integrity of vote data if nodes are run by a single entity. Distributing nodes among different government stakeholders can reduce this negative impact. The PoA algorithm is the natural choice for a permissioned network, as PoW and PoS only add redundant complexity to the network if it's permissioned.

Therefore it can be stated, under the limitations of the design, a setup of 14 validator nodes using the QBFT consensus algorithm maximizes the decentralization of

the network under the constraint of maintaining an approximation of Besu's baseline performance, as outlined in the literature review.

3.2.2 Additional parameters

Typically, gas fees are required for executing transactions. However, in this design, such fees introduce unnecessary complexity, as the validator nodes are permissioned and do not require incentives for validating transactions. Furthermore, the design does not rely on tokens for any functionality within the voting process; instead, all logic is managed by the smart contracts. As a result, setting gas fees for transactions to zero was deemed appropriate for this context.

An alternative approach to prevent DDoS attacks or unauthorized use of the blockchain, such as deploying unrelated smart contracts, could involve distributing gas to those with voting privileges. However, this idea was dismissed due to concerns that the added complexity in the smart contracts might impede the network's performance in comparison with the alternative. Instead of compromising between security and performance, utilizing a transaction filter to restrict transactions solely to the voting smart contract offers a more effective solution. This can be achieved by implementing the public interface "TransactionFilter" and specifying the allowed addresses as those of the deployed voting smart contracts. [21] Further mitigation of DDoS attacks from spamming the voting smart contract could be achieved with a ratelimiter.

In summary, the recommended blockchain design for this application should support the EVM, be public to maintain transparency, permissioned to adhere to the limitations and curtail risks and use approximately 14 nodes running a PoA algorithm (QBFT) to maximize for performance and decentralization.

3.3 Smart Contract specifications

As previously stated, the smart contract will be written in the Solidity programming language as a consequence of the blockchain utilizing the EVM. The design goal is to create a polling system, where there can be multiple polls open at the same time, consequently only needing one smart contract for all polls, instead of one for each poll, reducing the administration complexity. In order to provide similar functionalities to traditional voting systems, the smart contract meet the following specification:

- *Admin configurations*
  - (i) Ability to set a vote period at start of poll
  - (ii) Ability to delegate voting rights
  - (iii) Ability to revoke voting rights
  - (iv) Ability to add options before poll start
  - (v) Ability to create multiple polls
  - (vi) Ability to remove polls
  - (vii) Ability to use a ratelimiter
- *Integrity*
  - (i) Options are immutable after poll started
  - (ii) Only votes cast during the poll period are recorded
  - (iii) Ability to change their vote during the poll period
  - (iv) Only eligible voters can cast a vote
  - (v) Only one vote per person per poll
- *Transparency*

(i) Publicly independently available poll results

(ii) Receipt of vote

The decision to deviate from the receipt-free principle, commonly adopted to prevent vote buying and coercion, was motivated by the introduction of a vote-changing mechanism. When voters can modify their cast votes, the evidence of a particular vote becomes unreliable, as it can be altered at any point during the voting period. This reasoning also applies to coercion, which would only be effective if the vote was monitored until the end of the polling period.

In essence, the added transparency gained from enabling voters to verify that their votes were recorded as intended is considered more valuable than strictly adhering to the receipt-free principle. By incorporating the ability to change votes, the risks associated with vote buying and coercion are reduced to a level that could be regarded as edge cases, making the trade-off worthwhile.

Chapter 4

Implementation

In this chapter, the implementation details of the Polling System Smart Contract designed in the previous chapter. We will begin by explaining the structure of the contract and then explain the functions and modifiers that are part of the contract. The contract is written in the Solidity language, version 0.8.0 and makes use of a library Secp256k1.sol [22] that provides cryptographic primitives used for verifying blind signatures. The full source code is available in appendix A.

4.1 The smart contract

The polling system contract uses several structures, mappings, and functions, along with event emitters for recording specific actions and constants for ECC operations. The contract includes Voter, Option, and Poll structures that serve as the primary building blocks for the contract, as seen in listing on the next page. The contract also contains modifier functions that enforce specific restrictions on function execution.

The constructor sets the contract deployer as the admin and takes the public key of the poll admin in raw affine coordinates. This implementation grants the deployer control over the admin functions and stores the public key on-chain.

```
struct Voter {
        bool voteRight;
        mapping(uint => uint) vote;
        mapping(uint => bool) hasVoted; //pollId => bool
        uint lastReq;
    }

    struct Option {
        uint id;
        string name;
        uint voteCount;
    }

    struct Poll {
        uint id;
        uint pollStart;
        uint pollEnd;
        Option[] options;
    }

    address public admin;
    uint public pollCounter;
    mapping(uint => Poll) public polls;
    mapping(address => Voter) public voters;
    bool ratelimiterEnabled;
    uint RATELIMIT;     //unix time

    event VoteCast(uint indexed pollId, address indexed
```

```
    voter, uint indexed option);
event PollCreated(uint indexed pollId, uint startTime,
    uint endTime);
event PollRemoved(uint indexed pollId);

uint256[3] public pubKeyOfOrganizer;
//constants used in ECC operations
uint256[3] public generatorPoint;
uint constant n = 0
    xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141
    ;
uint constant p = 0
    xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
    ;

constructor(uint[2] memory pubKey) {
    admin = msg.sender;

    //(pubKeyOfOrganizer[0], pubKeyOfOrganizer[1]) =
        affine(x, y) of the organizers public key
    pubKeyOfOrganizer[0] = pubKey[0];
    pubKeyOfOrganizer[1] = pubKey[1];
    pubKeyOfOrganizer[2] = 1;

    generatorPoint[0] = 0
        x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798
        ;
    generatorPoint[1] = 0
        x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8
```

```
            ;
        generatorPoint[2] = 1;
    }
```

Listing 4.1: Program structs and state variables

4.1.1 Access control

The contract employs three modifier functions to regulate access to functions:

(i) `onlyAdmin`: Restricts access to admin-only functions, i.e the deployer of the contract.

(ii) `onlyDuringPollPeriod`: Ensures that voting can only take place within the specified time range of a poll.

(iii) `ratelimiter`: Prevents network spam and abuse by limiting the rate at which users can call certain functions.

```
modifier onlyAdmin(){
    require(msg.sender == admin, "Not admin");
    _;
}
modifier onlyDuringPollPeriod(uint pollId) {
    require(block.timestamp >= polls[pollId].pollStart &&
       block.timestamp <= polls[pollId].pollEnd,
            "Cannot vote outside of poll period");
    _;
}
modifier ratelimiter(address sender){
    if(ratelimiterEnabled){
```

```
        require(block.timestamp >= voters[sender].lastReq +
            RATELIMIT, "Ratelimited: Too many requests");
        voters[sender].lastReq = block.timestamp;
    }_;
}
```

Listing 4.2: The modifiers

4.1.1.1 Admin Functions

Admin functions allow the contract deployer to manage and control the voting process. These functions include:

(i) `ratelimiterSwitch`: Enables or disables the ratelimiter.

(ii) `changeRatelimit`: Changes the rate at which someone can send transactions to the contract by modifying the RATELIMIT value.

(iii) `createPoll`: Creates a new poll with specified options passed as a string array, start time and end time of the poll formatted unix epoch time.

(iv) `removePoll`: Removes an existing poll.

(v) `revokeVotingRight`: Revokes the voting right from a specified address.

The implementation of the admin functions can be seen in the listing below.

```
function ratelimiterSwitch(bool enabled) public onlyAdmin{
        ratelimiterEnabled = enabled;
    }

    function changeRatelimit(uint rate) public onlyAdmin{
        RATELIMIT = rate;
```

```
    }

    function createPoll(string[] memory optionNames, uint
       startTime, uint endTime) public onlyAdmin {
        Poll storage newPoll = polls[pollCounter];
        newPoll.id = pollCounter++;
        newPoll.pollStart = startTime;
        newPoll.pollEnd = endTime;

        for(uint i = 0; i < optionNames.length; i++) {
            newPoll.options.push(Option(i, optionNames[i],
               0));
        }
        emit PollCreated(newPoll.id, startTime, endTime);
    }

    function removePoll(uint pollId) public onlyAdmin {
        require(polls[pollId].id == pollId, "Invalid poll ID
           ");
        delete polls[pollId];
        emit PollRemoved(pollId);
    }

    function revokeVotingRight(address voter) public
       onlyAdmin{
        voters[voter].voteRight = false;
    }
```

Listing 4.3: The admin functions

4.1.1.2 User Functions

User functions enable the voters to interact with the polls and vote on options, as well as fetching the vote tally and seeing their own vote.

(i) `getVotingRights`: Validates a blind signature provided by a voter, checks if the signature was made by the private key corresponding to the poll organizers public key stored in the contract and if it was the senders address that was signed. If valid, assigns the right to vote to the address of the sender.
This is done by reconstructing the signature from the senders address, the public key of the poll organizers and the blind signature, comparing the result to the blind signature. See the outlined scheme for more details.

```
function getVotingRights(uint256 c, uint256 s) public
   ratelimiter(msg.sender) {
    string memory voter = Strings.toHexString(uint256(
       uint160(msg.sender)), 20);

    uint[3] memory cP = Secp256k1.multiply(c,
       pubKeyOfOrganizer);
    uint[3] memory sG = Secp256k1.multiply(s,
       generatorPoint);
    uint[3] memory sum = Secp256k1.jacobianAdd(cP, sG);
       //convert the x coordinate from Jacobian to
       affine coordinate before projecting the x
       coordinate
    uint[2] memory affineSum = Secp256k1.
       jacobianToAffine(sum);
    uint projection = affineSum[0] % n;
```

```
    uint cVerify = uint(keccak256(abi.encodePacked(voter
        , Secp256k1.uintToString(projection))));
    require(c == cVerify, "Invalid Signature");
    voters[msg.sender].voteRight = true;
}
```

Listing 4.4: Source code of the getVotingRights function

(ii) `vote`: Allows voters to cast their votes on an option in a specific poll, if they have already voted, their previous vote will be deducted from that option's vote count and added to their new choice.

```
function vote(uint pollId, uint option) public
   onlyDuringPollPeriod(pollId) ratelimiter(msg.sender){
    Voter storage sender = voters[msg.sender];

    require(sender.voteRight, "Has no right to vote");
    require(polls[pollId].id == pollId, "Invalid poll ID
       ");
    require(option >= 0 && option< polls[pollId].options
       .length, "Invalid optionID");

    //if they have already voted, remove prior vote
       before adding new one
    if(sender.hasVoted[pollId]){
        polls[pollId].options[sender.vote[pollId]].
           voteCount -= 1;
    }

    sender.hasVoted[pollId] = true;
    sender.vote[pollId] = option;
    polls[pollId].options[option].voteCount += 1;
    emit VoteCast(pollId, msg.sender, option);
}
```

Listing 4.5: Source code of the vote function

(iii) `voteTally`: Displays the tally of votes for each option in a specific poll, returned as two arrays, one containing the options and the other their respective vote counts. Index zero of the vote count array has the vote count for the option with index zero in the options array, and so forth.

```
function voteTally(uint pollId) public ratelimiter(msg.
   sender)
        returns (string[] memory optionNames_,
                uint[] memory voteCounts_) {

        require(polls[pollId].id == pollId, "Invalid
           poll ID");
        Poll memory poll = polls[pollId];
        uint pollSize = poll.options.length;
        voteCounts_ = new uint[](pollSize);
        optionNames_ = new string[](pollSize);

        for(uint i = 0; i < pollSize; i++){
            optionNames_[i] = poll.options[i].name;
            voteCounts_[i] = poll.options[i].voteCount;
        }
}
```

Listing 4.6: Source code of the voteTally function

The function returns two arrays, one for the options and the other for their corresponding vote counts. This approach was chosen because it reduces the number of transactions required to verify the data as opposed to having separate getter functions for options and vote counts. By reducing the transaction count, this implementation can improve the efficiency and scalability of the voting system.

4.1.2 Poll data

While it's true that transactions can be tracked through the transaction history or logs of a blockchain, events provide some advantages over directly examining the transaction history, as it makes the data more structured and allows for easier interfacing. The contract emits events to log the polling process to make it easier to audit and integrate with other systems.The events include:

(i) `VoteCast`: Triggered when a vote is cast.

(ii) `PollCreated`: Triggered when a new poll is created.

(iii) `PollRemoved`: Triggered when a poll is removed.

```
event VoteCast(uint indexed pollId, address indexed voter,
   uint indexed option);
event PollCreated(uint indexed pollId, uint startTime, uint
   endTime);
event PollRemoved(uint indexed pollId);
```

Listing 4.7: The events

4.1.3 Contract specification implementation

This section describes the implementation details of the given Solidity smart contract PollingSystem in relation to the design specifications provided in the design chapter.

- *Admin configurations*

  (i) **Ability to set a vote period at the start of the poll:** The `createPoll` function allows the admin to set the `startTime` and `endTime` for each poll created.

  (ii) **Ability to delegate voting rights:** The `getVotingRights` function allows the admin to grant voting rights to a specific address.

  (iii) **Ability to revoke voting rights:** The `revokeVotingRight` function allows the admin to revoke voting rights from a specific address.

  (iv) **Ability to add options before poll start:** The `createPoll` function enables the admin to add multiple options using the `optionNames` array parameter.

  (v) **Ability to create multiple polls:** The `createPoll` function can be called multiple times to create new polls, incrementing the `pollCounter`.

  (vi) **Ability to remove polls:** The `removePoll` function allows the admin to remove a poll by providing a valid `pollId`.

  (vii) **Ability to use a ratelimiter:** The `ratelimiterSwitch` and `changeRatelimit` functions allow the admin to enable, disable, and adjust the ratelimiter.

- *Integrity*

  (i) **Options are immutable after the poll started:** Once a poll has been created, its options cannot be changed since it's stored on-chain and no ability to change a poll is implemented

(ii) **Only votes cast during the poll period are recorded:** The `onlyDuringPollPeriod` modifier in the `vote` function ensures that votes can only be cast within the specified period.

(iii) **Ability to change their vote during the poll period:** The `vote` function allows voters to change their vote while the poll is active. The voter simply votes again, their previous vote is deducted and the new one is added.

(iv) **Only eligible voters can cast a vote:** The `vote` function checks if the voter has the right to vote using the `require(sender.voteRight, "Has no right to vote")` statement.

(v) **Only one vote per person per poll:** The `vote` function checks if the voter has already voted, and if so, removes the previous vote before adding the new one.

- *Transparency*

(i) **Publicly independently available poll results:** The `voteTally` function returns an array of proposal names and their respective vote counts, making the poll results publicly available.

4.2 The voting process

The following subsections describe how the design of the voting process was implemented.

4.2.1 Authentication

- *Generating and Distributing Unique Identifiers:* The government generates unique account details with a random number generator or some other method and arbitrarily assigns them to people with voting rights before securely sending them through mail to each eligible citizen. Only the account details are stored on the government servers database, no personal information.

4.2.2 Distribution of voting rights

- *Wallet Creation and management:*

  (i) The first time the voter logs in, they are prompted to setup their account before they can vote. The client generates a wallet with the javascript library web3.js. The voter creates a secret password, which will not be saved. This wallet will be used to cast votes.

  (ii) The wallet is encrypted with their secret password using a symmetric encryption algorithm (AES), before it's sent for storage on the server. This is better than storing the wallet in the browser, the voter can use any device and not risk losing the wallet when clearing browser data and arguably more secure, provided the password is strong.

- *Government Approval Process:*

  (i) The voter's wallet address is blinded according to the blind signature scheme implementation in javascript, before being sent to the server

(ii) The server checks the account details in the database, if it's valid and the user hasn't received voting rights before, the server uses its private key - which is tied to the public key stored in the smart contract - to sign the blinded version of the voters wallet address. This signed message is sent back to the voter.

- *Unblinding and Smart Contract whitelisting:*

(i) The client uses their blinding factor to unblind the signed address.

(ii) The client sends a transaction to the smart contract function `getVotingRights` from the voter's newly generated wallet. This transaction includes the blind signature.

(iii) In the `getVotingRights` function, the smart contract takes the blind signature as input. It then uses the address of the sender (i.e the voters wallet address), along with the poll organizers public key stored in the contract to rebuild the signature. If the rebuilt signature matches the signature provided to the function by the voter, the contract then delegates voting rights to the voters's address by storing the mapping of the voters's address to a Voter struct with a `voteRight` field set to true.

4.2.3 Casting the Vote

(i) When the voter logs in to the website, the server sends the encrypted wallet to the voters browser.

(ii) They cast their vote by choosing an option in an active poll, entering their secret password, which then decrypts their wallet with voting rights, before using it to send the vote transaction to the smart contract.

In this process, the government never has any direct ties between the voter's wallet address and the government distributed accounts. It only sees the blinded version of the wallet address, which it can't link to the actual address because the blinding factor is only known within the client.

In this chapter, we have discussed the implementation details of the voting process, the Smart Contract and the client-server interaction. The contract is designed to provide a secure and transparent voting process for government polls. The contract structure, along with the various functions and modifiers, ensures that the design specifications were met, while the incorporation of blind signatures in the distribution of voting rights ensures the confidentiality of the voters wallet addresses.

Chapter 5

Results

5.1 The voting process

In this section, we evaluate the proposed blockchain-based e-voting system's design by analyzing the extent to which it meets the requirements outlined in the methods section. These requirements are confidentiality, auditability, contestability, decentralization, and performance. We also discuss the implementation of the smart contract in the context of these requirements.

(i) *Confidentiality*

The proposed e-voting system aims to maintain voter confidentiality by ensuring that each vote is cast anonymously. This is achieved through the generation of anonymous wallets within the voter's browser and subsequently encrypting the wallet using the voter's secret password before storing it on government servers. The distribution of voting rights utilizes blind signatures to hide the voters wallet address. This prevents the association of the government-issued account details with the voter's address, which will be publicly visible on the blockchain once they cast their vote. Even if malicious actors within the government were to

record or leak the account details and the identities they were distributed to, it would not compromise the integrity of the vote, as these details are not linked to the voting addresses.

The risk of storing the encrypted wallet on the servers, is in the case of malicious actors getting into the database and trying to brute force the password. If the passwords are not strong, there's a risk of associating the website account details to the address used for voting. This risk is somewhat mitigated by the fact that the website account details are already pseudonymous via the distribution scheme of these addresses. Only an internal malicious actor that was involved in the distribution of these website accounts would be able to directly tie peoples identity to their voting addresses.

Other external malicious actors would have to not only compromise the database, but also analyse the network traffic to determine the IP addresses of the users and try to identify them by collaborating with or coercing ISPs. Alternatively using some kind of geolocation service that map IP addresses to physical locations.

Another attack vector against voter confidentiality is timing attacks. If there are only a few or no addresses being whitelisted between the blind signing of the address and the whitelisting in the contract, one might be able to connect the website account that requested the blind signature to the address that sends the transaction to the `getVotingRights` function for whitelisting.

The network risks might be mitigated by using the Tor network, while the risk for timing attacks can be reduced by introducing some random element to the timing of the call to the contract for whitelisting after a signature.

As a result, the system upholds the principle of the secret ballot and protects voter confidentiality to some degree.

(ii) *Auditability*

The e-voting system allows for public independent auditing by employing a public permissioned blockchain. This configuration enables the general public to independently read and verify the vote data on the blockchain, ensuring transparency in the voting process while maintaining a robust baseline of protection against network takeover. The system also adheres to the limitation of it running on national servers, due to the fact that the chain is permissioned. Moreover, the smart contract implementation supports vote tallying by allowing users to retrieve proposal names and their vote counts for each poll, further enhancing the accessability for the general public to audit and verify the vote data independently.

(iii) *Contestability*

The system is contestable through the transparent and tamper-resistant nature of the blockchain. By utilizing a public blockchain, the voting data is accessible to the public, enabling an archiving of the data. If at any time the majority of the validators in the network were to try revert transaction and change previous blocks, it would be detectable when comparing with the archived copies of the previous blocks by archive nodes run by the public. When an error is detected, the public can verify the evidence and contest the results, ensuring the integrity of the voting process by the contestable property of the system.

(iv) *Decentralization*

The proposed e-voting system aims to achieve a level of decentralization by distributing validators between different stakeholders of the system. The decentralization might be the weakest of all the aspects in this design, which is a consequence of the requirement that the application must be on national servers, consequently leading to a permissioned blockchain. The public permissioned blockchain configuration enables only the validators, who run nodes on national servers, to participate in the consensus process. Although this design does not

provide complete decentralization, it strikes a balance between security, transparency, and control while adhering to the constraints outlined in the methods and limitations sections of the thesis.

(v) *Performance*

The performance of the e-voting system is not tested in practice but is addressed through the choice of a public permissioned blockchain. This configuration allows for efficient communication and resource allocation among the validator nodes, resulting in a more reliable network with better performance. Additionally, the smart contract is designed to handle multiple polls and options, ensuring that the system can scale to accommodate various use cases. However, as the system is only a PoC, further testing and optimization may be necessary to determine the system's performance.

5.2 Smart Contract Evaluation

The smart contract was evaluated through unit testing and deployment within a VM environment using the Remix IDE as well as being deployed on the Sepolia testnet. All functions underwent unit testing, with the exception of the `vote` and `delegateVotingRight` functions due to challenges encountered in making transactions from custom addresses within the Remix unit tester. After trying NATSPEC comments to make transactions with custom `msg.sender` and making low level calls from other addresses without any reliable results, it was decided to test the vote function along with all time dependent functionalities within a live testing environment involving multiple addresses.

The unit tests, showcased in Appendix A, confirm that the tested functions perform as expected. Since Solidity is a statically-typed language, edge cases involving unexpected data types result in compile-time errors, one cannot include these types of tests in the unit testing. Additionally, time-dependent functionalities, such as the

"onlyDuringPollPeriod" and "ratelimiter" modifiers, could not be conveniently tested within the unit testing environment due to it not being live and the absence of sleep functions or other time-related control mechanisms. As a result, these functionalities were assessed within the VM environment.

The integration testing with the website was done with the help of metamask and web3.js. The choice to use metamask was made due to the need for testnet ETH to send transactions and test functions in the contract. Since the design utilizes a premissioned blockchain with no fees, using the websites natively generated wallets, as intended would introduce unecessary complexity to the testing without adding any strength to the test results. The contracts where deployed from the address in metamask after it had recieved testnet tokens. Multiple transactions were sent to the `getVotingRights` function, both with valid and invalid blind signatures that were generated on the website for the address in the metamask wallet. The function worked as expected, reverting on false inputs or signatures, while delegating voting rights whenever the signature was valid for that address.

The `vote` function and the `createPoll` function were also tested. Once three polls were created, they were properly displayed on the website by utilizing the getter functions in the contract. The voting was also recorded as expected.

The proposed blockchain-based e-voting system was evaluated based on the requirements of confidentiality, auditability, contestability, decentralization, and performance. The system maintains voter confidentiality by generating wallets in the voter's browser before delegating voting rights to them with the use of blind signatures and encrypting them before storing them on government servers. The design employs a public permissioned blockchain to allow public independent auditing and ensures contestability by the inherent tamper-resistant property of the distributed ledger that blockchain offers. While the system is not completely decentralized due to the requirement of running on

national servers, it strikes a balance between security, transparency, and control. The smart contract was evaluated outside of its suggested infrastructure design, but within it's intended execution engine (the EVM). The contract was unit tested within a VM and deployed on a testnet for integration testing with the website. The results show that the contract functions as expected.

Chapter 6

Conclusions

6.1 Discussion

The integration of a government-issued digital ID system could enhance the system's effectiveness and user-friendliness. This would simplify the distribution of voter rights, reducing the cost and complexity of the logistics involved.

The permissioned blockchain configuration, while limiting in terms of decentralization, allows for more control over the network, satisfying the requirement of running on national servers. Although the level of decentralization achieved is not optimal and might be a point of critique and potential vulnerability, it offers a more transparent alternative to a regular database while maintaining security, and control of the network. The system's performance, while theoretically sound, is yet to be tested under real-world conditions, which is a necessary step to confirm its feasibility and reliability.

6.2 Conclusion

The thesis successfully designs, partially implements and evaluates a PoC of a blockchain-based e-voting system that satisfies the key requirements of confidentiality, auditability, contestability, and decentralization to a certain degree. Additionally, it adheres to the requirement of hosting the application on national servers and without the use of digital IDs.

The system's design allows for public auditing, promoting trust in the e-voting process. Despite limitations regarding decentralization and untested real-world performance, the system meets the requirements of an evidence based voting system to a significant degree.

6.3 Future work

The full stack implementation of this design would strengthen the confidence in the concept. More comprehensive testing of the system under real-world conditions would provide insights into its performance and reliability. This could involve simulating large-scale polls with numerous participants and verifying the system's ability to handle a realistic network load.

Secondly, while the system meets the requirement of running on national servers, it is worth exploring how this could be adapted to achieve greater decentralization without compromising security and control to a level uncomfortable for the stakeholders. This could involve investigating other blockchain configurations or consensus mechanisms that may offer a better balance between these requirements.

Bibliography

[1] wikipedia.org. 2000 united states presidential election in florida.

[2] wikipedia.org. Dominion voting systems - software.

[3] stanford.edu. The history of cryptography, 2023.

[4] Simanta Shekhar Sarmah. Understanding blockchain technology. *Computer Science and Engineering*, 8(2):23–29, 2018.

[5] hashcash.org. Hashcash, 2003.

[6] wikipedia.org. Hashcash, 2023.

[7] investopedia.com. B-money, 2023.

[8] coinmarketcap.com. Historical snapshots, 2023.

[9] Xing Fan, Baoning Niu, and Zhenliang Liu. Scalable blockchain storage systems: research progress and models. *Computing*, 104(6):1497–1524, 2022.

[10] Fran Casino, Thomas K. Dasaklis, and Constantinos Patsakis. A systematic literature review of blockchain-based applications: Current status, classification and open issues. *Telematics and Informatics*, 36:55–81, 2019.

[11] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.", 2014.

[12] Paolo Tasca and Claudio J Tessone. Taxonomy of blockchain technologies. principles of identification and classification. *arXiv preprint arXiv:1708.04872*, 2017.

[13] ethereum.org. Intro to ethereum.

[14] Cuneyt Gurcan Akcora, Yulia R Gel, and Murat Kantarcioglu. Blockchain networks: Data structures of bitcoin, monero, zcash, ethereum, ripple, and iota. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 12(1):e1436, 2022.

[15] Friðrik Þ Hjálmarsson, Gunnlaugur K Hreiðarsson, Mohammad Hamdaqa, and Gísli Hjálmtỳsson. Blockchain-based e-voting system. In *2018 IEEE 11th international conference on cloud computing (CLOUD)*, pages 983–986. IEEE, 2018.

[16] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. A smart contract for boardroom voting with maximum voter privacy. In *Financial Cryptography and Data Security: 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers 21*, pages 357–375. Springer, 2017.

[17] Ruhi Taş and Ömer Özgür Tanrıöver. A systematic review of challenges and opportunities of blockchain for e-voting. *Symmetry*, 12(8):1328, 2020.

[18] Caixiang Fan, Changyuan Lin, Hamzeh Khazaei, and Petr Musilek. Performance analysis of hyperledger besu in private blockchain. In *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 64–73. IEEE, 2022.

[19] Rådata och statistik, Apr 2023.

[20] GAVIN WOOD. Ethereum: A secure decentralised generalised transaction ledger berlin version, Oct 2022.

[21] Hyperledger. Besu/transactionfilter.java at main ů hyperledger/besu.

[22] William J Buchanan. Elliptic curve operations within a solidity contract. `https://asecuritysite.com/ecc/ethereum16`, 2023. Accessed: September 11, 2023.

Bilagor

Appendix A

Appendix A

...

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/utils/Strings.sol";
import {Secp256k1} from "./Secp256k1.sol";

contract PollingSystem {
    struct Voter {
        bool voteRight;
        mapping(uint => uint) vote;
        mapping(uint => bool) hasVoted; //pollId => bool
        uint lastReq;
    }

    struct Option {
        uint id;
        string name;
```

```
        uint voteCount;
    }

    struct Poll {
        uint id;
        uint pollStart;
        uint pollEnd;
        Option[] options;
    }

    address public admin;
    uint public pollCounter;
    mapping(uint => Poll) public polls;
    mapping(address => Voter) public voters;
    bool ratelimiterEnabled;
    uint RATELIMIT;     //unix time

    event VoteCast(uint indexed pollId, address indexed
       voter, uint indexed option);
    event PollCreated(uint indexed pollId, uint startTime,
       uint endTime);
    event PollRemoved(uint indexed pollId);

    uint256[3] public pubKeyOfOrganizer;
    //constants used in ECC operations
    uint256[3] public generatorPoint;
    uint constant n = 0
       xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD036414
       ;
```

```
uint constant p = 0
   xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
   ;

constructor(uint[2] memory pubKey) {
    admin = msg.sender;

    //(pubKeyOfOrganizer[0], pubKeyOfOrganizer[1]) =
       affine(x, y) of the organizers public key
    pubKeyOfOrganizer[0] = pubKey[0];
    pubKeyOfOrganizer[1] = pubKey[1];
    pubKeyOfOrganizer[2] = 1;

    generatorPoint[0] = 0
       x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798
       ;
    generatorPoint[1] = 0
       x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8
       ;
    generatorPoint[2] = 1;
}

modifier onlyAdmin(){
    require(msg.sender == admin, "Not admin");
    _;
}

modifier onlyDuringPollPeriod(uint pollId) {
    require(block.timestamp >= polls[pollId].pollStart
```

```
        && block.timestamp <= polls[pollId].pollEnd,
             "Cannot vote outside of poll period");
    _;
}

modifier ratelimiter(address sender){
    if(ratelimiterEnabled){
        require(block.timestamp >= voters[sender].
           lastReq + RATELIMIT, "Ratelimited: Too many
           requests");
        voters[sender].lastReq = block.timestamp;
    }
    _;
}

//Admin functions
function ratelimiterSwitch(bool enabled) public
   onlyAdmin{
    ratelimiterEnabled = enabled;
}

function changeRatelimit(uint rate) public onlyAdmin{
    RATELIMIT = rate;
}

function createPoll(string[] memory optionNames, uint
   startTime, uint endTime) public onlyAdmin {
    Poll storage newPoll = polls[pollCounter];
    newPoll.id = pollCounter++;
```

```
        newPoll.pollStart = startTime;
        newPoll.pollEnd = endTime;

        for(uint i = 0; i < optionNames.length; i++) {
            newPoll.options.push(Option(i, optionNames[i],
               0));
        }
        emit PollCreated(newPoll.id, startTime, endTime);
    }

    function removePoll(uint pollId) public onlyAdmin {
        require(polls[pollId].id == pollId, "Invalid poll ID
           ");
        delete polls[pollId];
        emit PollRemoved(pollId);
    }

    function revokeVotingRight(address voter) public
       onlyAdmin{
        voters[voter].voteRight = false;
    }

    //User functions
    function getVotingRights(uint256 c, uint256 s) public
       ratelimiter(msg.sender) {
        string memory voter = Strings.toHexString(uint256(
           uint160(msg.sender)), 20);

        uint[3] memory cP = Secp256k1.multiply(c,
```

```
        pubKeyOfOrganizer);
    uint[3] memory sG = Secp256k1.multiply(s,
        generatorPoint);
    uint[3] memory sum = Secp256k1.jacobianAdd(cP, sG);
        //convert the x coordinate from Jacobian to
        affine coordinate before projecting the x
        coordinate
    uint[2] memory affineSum = Secp256k1.
        jacobianToAffine(sum);
    uint projection = affineSum[0] % n;

    uint cVerify = uint(keccak256(abi.encodePacked(voter
        , Secp256k1.uintToString(projection))));
    require(c == cVerify, "Invalid Signature");
    voters[msg.sender].voteRight = true;
}

function vote(uint pollId, uint option) public
    onlyDuringPollPeriod(pollId) ratelimiter(msg.sender){
    Voter storage sender = voters[msg.sender];

    require(sender.voteRight, "Has no right to vote");
    require(polls[pollId].id == pollId, "Invalid poll ID
        ");
    require(option >= 0 && option< polls[pollId].options
        .length, "Invalid optionID");

    //if they have already voted, remove prior vote
        before adding new one
```

```
        if(sender.hasVoted[pollId]){
            polls[pollId].options[sender.vote[pollId]].
               voteCount -= 1;
        }

        sender.hasVoted[pollId] = true;
        sender.vote[pollId] = option;
        polls[pollId].options[option].voteCount += 1;
        emit VoteCast(pollId, msg.sender, option);
    }

    function voteTally(uint pollId) public ratelimiter(msg.
       sender) returns (string[] memory optionNames_, uint[]
        memory voteCounts_) {
        require(polls[pollId].id == pollId, "Invalid poll ID
           ");
        Poll memory poll = polls[pollId];
        uint pollSize = poll.options.length;
        voteCounts_ = new uint[](pollSize);
        optionNames_ = new string[](pollSize);

        for(uint i = 0; i < pollSize; i++){
            optionNames_[i] = poll.options[i].name;
            voteCounts_[i] = poll.options[i].voteCount;
        }
    }
}
```

Listing A.1: Full source code of the smart contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

import "./EllipticCurve.sol";

library Secp256k1 {

    uint256 public constant GX = 0
       x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F8179
       ;
    uint256 public constant GY = 0
       x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B
       ;
    uint256 public constant AA = 0;
    uint256 public constant BB = 7;
    uint256 public constant PP = 0
       xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2
       ;

    /// @dev Multiply point (x, y, z) times d.
    /// @param _d scalar to multiply
    /// @param _P jacobian point
    /// @return (qx, qy, qz) d*P in Jacobian
    function multiply(uint256 _d, uint256[3] memory _P) pure
        public returns (uint256[3] memory){
        uint256[3] memory point;
        (point[0], point[1], point[2]) = EllipticCurve.
           jacMul(_d, _P[0], _P[1], _P[2], AA, PP);
```

```
    return point;
}

function jacobianAdd(uint256[3] memory _p1, uint256[3]
   memory _p2) pure public returns (uint256[3] memory){
    uint256[3] memory sum;
    (sum[0], sum[1], sum[2]) = EllipticCurve.jacAdd(_p1
       [0], _p1[1], _p1[2],_p2[0], _p2[1], _p2[2], PP);
    return sum;
}

function jacobianToAffine(uint256[3] memory _p) pure
   public returns (uint256[2] memory){
    uint256[2] memory affineP;
    (affineP[0], affineP[1]) = EllipticCurve.toAffine(_p
       [0], _p[1], _p[2], PP);
    return affineP;
}

function uintToString(uint v) internal pure returns (
   string memory str) {
    uint maxlength = 78;
    bytes memory reversed = new bytes(maxlength);
    uint i = 0;
    while (v != 0) {
    uint remainder = v % 10;
    v = v / 10;
    reversed[i++] = bytes1(uint8(48 + remainder));
    }
```

```
        bytes memory s = new bytes(i);
        for (uint j = 0; j < i; j++) {
        s[j] = reversed[i - 1 - j];
        }
        str = string(s);
    }

    function add(uint256 x1, uint256 y1,uint256 x2, uint256
       y2 ) pure public returns (uint256, uint256) {
        return EllipticCurve.ecAdd(x1,y1,x2,y2,AA,PP);
    }

    function invMod(uint256 val, uint256 p) pure public
       returns (uint256){
        return EllipticCurve.invMod(val,p);
    }

    function expMod(uint256 val, uint256 e, uint256 p) pure
       public returns (uint256){
        return EllipticCurve.expMod(val,e,p);
    }

    function getY(uint8 prefix, uint256 x) pure public
       returns (uint256){
        return EllipticCurve.deriveY(prefix,x,AA,BB,PP);
    }

    function onCurve(uint256 x, uint256 y) pure public
       returns (bool){
```

```
        return EllipticCurve.isOnCurve(x,y,AA,BB,PP);
    }

    function inverse(uint256 x, uint256 y) pure public
       returns (uint256, uint256) {
        return EllipticCurve.ecInv(x,y,PP);
    }

    function subtract(uint256 x1, uint256 y1,uint256 x2,
       uint256 y2 ) pure public returns (uint256, uint256) {
        return EllipticCurve.ecSub(x1,y1,x2,y2,AA,PP);
    }

    function derivePubKey(uint256 privKey) pure public
       returns (uint256, uint256) {
        return EllipticCurve.ecMul(privKey,GX,GY,AA,PP);
    }

}
```

Listing A.2: The library Secp256k1.sol

```
// STEP 1 - Get R and P from server
const res1 = await fetch('/setup/generate-random-integer');
const data1 = await res1.json();

//R is a randomly selected point on the curve
const Rx = BigInteger.fromBuffer(data1.R.x);
const Ry = BigInteger.fromBuffer(data1.R.y);
const R = ecurve.Point.fromAffine(ecparams, Rx, Ry);
//organizers public key in raw format (affine point)
const Px = BigInteger.fromBuffer(data1.P.x);
const Py = BigInteger.fromBuffer(data1.P.y);
const P = ecurve.Point.fromAffine(ecparams, Px, Py);

console.log("R: " + R);
console.log("P: " + '(' + '0x' + toHex(P.affineX) + ', ' +
   '0x' + toHex(P.affineY));

/* STEP 2 - Blind the address
The requester randomly selects two integers  and   Zn,
   blinds the message, and then
calculates point A = kG +  G + P = (x, y), t = x (mod n). If
    t equals zero, then  and  should
be reselected. The requester calculates c = SHA3 (m || t), r
    = r  , where SHA3 is a
novel hash function computed with 32-bit words and c is the
   blinded message, and then sends
r to the signer.
*/
```

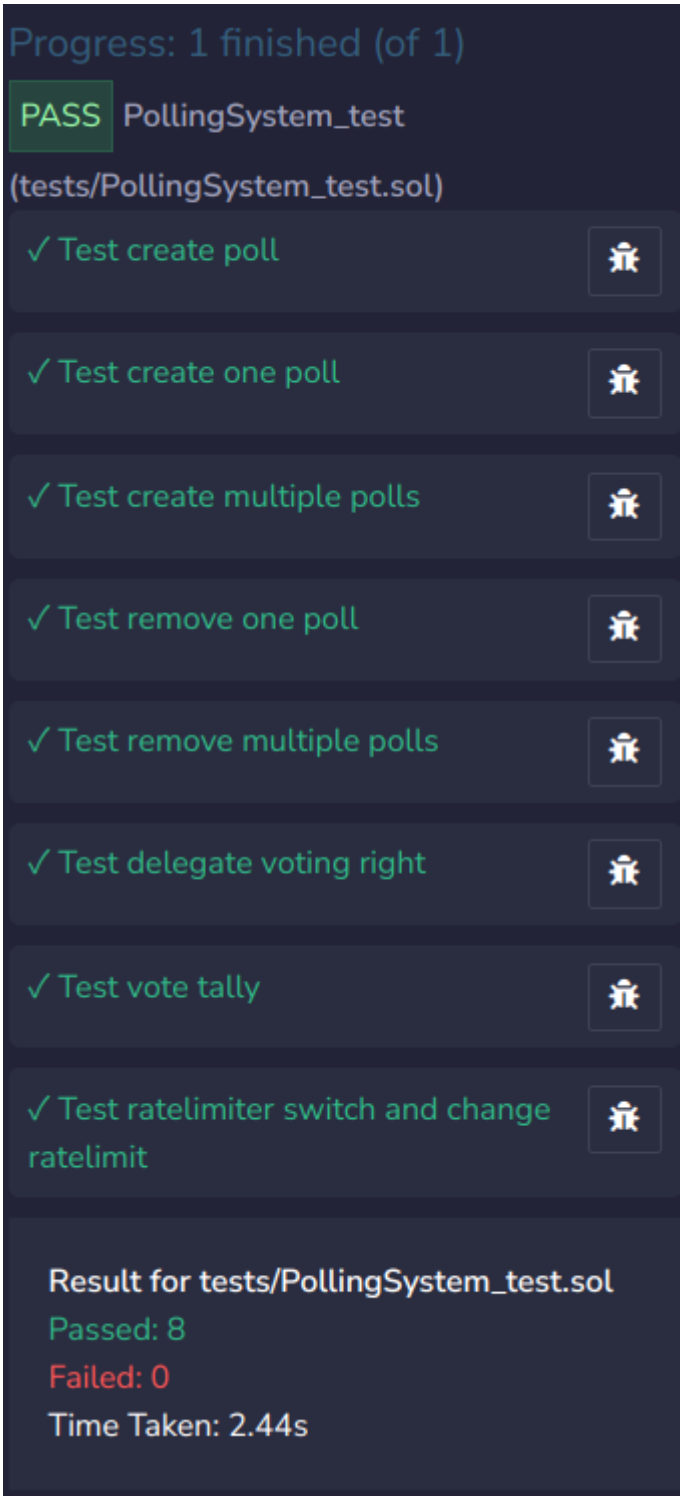

Figure A.1: Unit testing results

```
//due to tx cost, we use this address with eth balance to
   test
//the verification of the blind signature function on-chain
const voterAddress = "0
   x4231a4fe2e189932184aa42deb675aaad440e0c7";
//const voterAddress = voterWallet.address.toLowerCase();

const  = random(32);
const  = random(32);

const A = add(add(R,multiply(G,)),multiply(P,));
const t = A.x.mod(n).toString();

const r = BigInteger.fromHex(keccak256(voterAddress+t.
   toString()));
console.log("r:\n" +  '0x' + toHex(r));

const rBlind = r.subtract();
console.log("Blinded address (rBlind): " + '0x' + toHex(
   rBlind));
const res2 = await fetch('/setup/sign', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({"addressBlind": rBlind.toString()
       })
});
const data2 = await res2.json();
```

```
/* STEP 4 - Unblind the signature
The requester calculates s = s + , and (r, s) is the
   signature on m.
*/
const sBlind = BigInteger.fromBuffer(data2.sBlind);
console.log("sBlind: " + sBlind);

const s = sBlind.add().mod(n);
console.log("s: " + s);

/* STEP 5 - CLIENT
Both the requester and signer can verify the signature (r, s
   ) through the formation
r = SHA3(voterAddress || Rx(rP + sG) mod n)
*/
const toHash = add(multiply(P,r),multiply(G,s)).x.mod(n);
const r_verify = BigInteger.fromHex(keccak256(voterAddress+
   toHash).toString()).toString();

console.log("r_verify: ");
console.log('0x' + toHex(r_verify));

console.log("s: ");
console.log('0x' + toHex(s));

if (r == r_verify){
    console.log("Signature is valid");
    document.getElementById("step3").style.color = "green";
}else{
```

```
    console.log("Invalid Signature");
    document.getElementById("step3").style.color = "red";
    //handle invalid signature more
}

//Send signature (r, s) to smart contract from address for
   whitelisting
await verifyBlindSignature(r.toString(),s.toString());
```

Listing A.3: The blind signature scheme on the client-side

```
app.post('/setup/sign', ensureAuthenticated, noVotingRights,
    async (req, res) => {
    try {
        //3.1 - Sign blinded address
                const addressBlind = BigInteger.fromBuffer(
                   req.body.addressBlind);

                const k = BigInteger.fromBuffer(req.session.
                   k);
        if (!k)
                        return res.status(400).send("k not
                           found.");

        const sBlind = k.subtract(addressBlind.multiply(
           BigInteger.fromBuffer(privateKey)));

        delete req.session.k;
        //3.2. Update votingRights field in db to true
                await giveVotingRights(req.user.username);

        //3.3 Send signature
        res.json({"sBlind": sBlind.toString()});

    } catch (error) {
        console.error('Error during the signing process:',
           error);
        res.status(500).send('Internal Server Error');
    }
```

```
});
```

Listing A.4: The blind signature scheme on the server-side

Appendix B

Appendix B

...