
**BRNO UNIVERSITY OF TECHNOLOGY**
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**FACULTY OF INFORMATION TECHNOLOGY**
**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

**DEPARTMENT OF INTELLIGENT SYSTEMS**
**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**


# DECENTRALIZED E-VOTING ON SOLANA BLOCKCHAIN

**DECENTRALIZOVANÉ ELEKTRONICKÉ HLASOVANIE NA SOLANA BLOCKCHAINE**

**MASTER'S THESIS**
**DIPLOMOVÁ PRÁCE**


**AUTHOR** — **Bc. MARTIN HOŠALA**
**AUTOR PRÁCE**

**SUPERVISOR** — **Ing. IVAN HOMOLIAK, Ph.D.**
**VEDOUCÍ PRÁCE**



**BRNO 2021**

**Brno University of Technology**
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2021/2022

# Master's Thesis Specification


Student: **Hošala Martin, Bc.**
Programme: Information Technology and Artificial Intelligence
Specialization: Application Development
Title: **Decentralized E-Voting on Solana Blockchain**
Category: Security
Assignment:

1. Get familiar with principles of blockchains and smart contracts.
2. Study programming model of Solana blockchain and compare it with the programming model of Ethereum.
3. Get familiar with electronic voting on blockchain, in particular BBB-Voting and its scalable extension.
4. Analyze requirements of BBB-Voting and redesign it to be suitable for Solana programming model.
5. Implement BBB-Voting on Solana blockchain.
6. Test the performance of your implementation on a suitable case study.
7. Discuss the security of your implementation and further improvements.

Recommended literature:

- Venugopalan, Sarad, et al. "BBB-Voting: 1-out-of-k Blockchain-Based Boardroom Voting." *arXiv preprint arXiv:2010.09112* (2020).
- Anatoly Yakovenko: "Solana: A new architecture for a high performance blockchain v0.8.13",
- Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." *Ethereum project yellow paper* 151.2014 (2014): 1-32.

Requirements for the semestral defence:

- Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Homoliak Ivan, Ing., Ph.D.**
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2021
Submission deadline: May 18, 2022
Approval date: November 3, 2021

# Abstract

This work was aimed at finding out the suitability of Solana blockchain for BBB-Voting system and creating a prototype of this system based on the provided solutions on Ethereum. The problem with Ethereum is its performace – a larger voting would take weeks. Solana promises much higher performace. To create the final solution, it was necessary to analyze Solana's system, BBB-Voting, design BBB-Voting for Solana, implement and test it. The final prototype is implemented in Rust using the Anchor framework. During the development, it was found that the algorithm which within the BBB-Voting protocol is used for vote validation is too computationally intensive and therefore due to the current limit on Solana, the system cannot be deployed on the mainnet. However, it is expected that this limit will be changed and the system can be deployed in the future. In that case, a rough estimate of the speedup over Etherem counterparts is around 3000 %. The cost of voting on Solana is also an order of magnitude lower. As part of the work, a front-end for voting was also developed – a single-page web application built using ReactJS.



# Abstrakt

Táto práca bola zameraná na zistenie využiteľnosti Solana blockchainu pre hlasovací systém BBB-Voting a vytvorenie prototypu tohto systému na základe poskytnutých riešení pre Ethereum. Problém s Ethereom je jeho výkon – väčšie voľby by trvali týždne. Solana sľubuje omnoho vyšší výkon. Na vytvorenie výsledného riešenia bolo potrebné analyzovať systém Solana, BBB-Voting, navrhnúť BBB-Voting pre Solanu, implementovať a otestovať ho. Výsledný prototyp je implementovaný v jazyku Rust pomocou frameworku Anchor. Počas vývoja bolo zistené, že algoritmus, ktorý vrámci protokolu BBB-Voting slúži pre overovanie hlasov je príliš výpočetne náročný a preto kôli súčastnému limitu na Solane nie je možné systém nasadiť na mainnet. Avšak očakáva sa, že tento limit sa bude meniť a systém bude v budúcnosti môcť byť nasadený. V takom prípade sa hrubý odhad zrýchlenia oproti Etherovým náprotivkom pohybuje okolo 3000 %. Cena hlasovania na Solane je taktiež rádovo nižšia. Vrámci práce bol vyvinutý aj *front-end* pre hlasovanie – *single-page* webová aplikácia vytvorená pomocou ReactJS.



# Keywords

Decentralized voting, Blockchain, Smart contracts, BBB-Voting, Ethereum, Solana

# Klíčová slova

Decentralizované voľby, Blockchain, Smart kontrakty, BBB-Voting, Ethereum, Solana


# Reference


HOŠALA, Martin. *Decentralized E-Voting on Solana Blockchain.* Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Ivan Homoliak, Ph.D.

# Rozšířený abstrakt

Hlasovanie je neoddeliteľnou súčasťou demokratickej správy vecí verejných alebo správy vecí spoločných pre určitú skupinu. Oprávnení účastníci môžu hlasovať za svojho preferovaného kandidáta, politika alebo možnosť tajným hlasovaním. Výsledkom hlasovania je vyhlásenie víťazov na základe sčítania hlasov. Hlasovanie je zvyčajne centralizované, čo so sebou prináša mnohé negatívne dôsledky. Medzi ne patrí napríklad ľahká možnosť manipulácie s výsledkami, cenzúra alebo obmedzenie dostupnosti služby pre určitú skupinu ľudí.

Problémy centralizovaného hlasovania možno vyriešiť pomocou technológie blockchain, ktorá prevádzkuje decentralizovaný konsenzuálny protokol spoliehajúci sa na čestnú väčšinu všetkých konsenzuálnych uzlov. Blockchain je dnes rýchlo sa rozvíjajúca decentralizovaná technológia s veľkým potenciálom, ktorá sľubuje zlepšenie viacerých aspektov mnohých odvetví. Okrem iného technológia blockchain s platformami smart kontraktov poskytuje zaujímavé funkcie, ktoré možno využiť na riešenie súčasných problémov elektronického hlasovania.

V posledných rokoch boli navrhnuté rôzne systémy elektronického hlasovania založené na blockchaine. Jedým z nich je protokol BBB-Voting, ktorý vytvorili Ivan Homoliak z FIT VUT spolu s ďalšími tromi spolupracovníkmi. Jedná sa o hlasovací systém založený na blockchaine, ktorý umožňuje hlasovanie 1 z $k$, teda každý účastník môže hlasovať za jednu z k možností.

Protokol BBB-Voting má vynikajúce vlastnosti, ale keďže je vyvinutý pre Ethereum, skutočné voľby, ktoré by sa cez neho konali, sú obmedzené rýchlosťou tejto platformy. To nie je dobré, pretože v súčasnosti Ethereum poskytuje priepustnosť približne 16 tansakcií za sekundu. Pri BBB-Voting musí každý legitímny volič vykonať v sieti aspoň dve transakcie. Ak by teda išlo o veľké voľby s 1 000 000 účastníkmi, len na tieto voľby by bolo potrebných 37 hodín plného výkonu siete. Tieto voľby by teda v skutočnosti museli trvať niekoľko dní. Pri 10 000 000 voličoch by to trvalo týždne.

Táto práca sa zameriava na implementáciu BBB-Voting pre Solana blockchain, ktorý môže mať s dnešným hardvérom priepustnosť 710 000 transakcií za sekundu. BBB-Voting na blockchaine Solana je teda z hľadiska reálneho nasadenia pre veľké voľby oveľa sľubnejší ako na Ethereum blockchaine.

Praktická časť práce sa venuje vytváraniu výsledného funkčného prototypu na základe poznatkov z teoretickej časti. Pre jeho vytvorenie bolo potrebné zvoliť druh a navrhnúť spôsob jeho implementácie. Bol zvolený vývoj varianty využívajúcej aritmetiku celých čísel. Výsledný systém je implementovaný v jazyku Rust pomocou frameworku Anchor, čo je framework pre vývoj smart kontraktov na Solane. Ukázalo sa však, že algoritmus BBB-Voting je výpočtovo náročný a preto jeho nasadenie na Solana mainnet v súčastnosti nie je možné. Metóda výsledného prototypu slúžiaca pre hlasovanie totiž spotrebuje viac tzv. compute units ako je súčastný maximálny limit na Solane. Tento limit sa však môže časom zmeniť a systém bude potom využiteľný. V taktomto prípade je hrubé odhadované zlýchlenie systému oproti Ethereovému náprotivku okolo 3000 %. Takisto cena hlasovania je na Solane výrazne nižšia – pohybuje sa v tisícinách centu za hlas, kdežto na Ethereu sa jedná o jednotky až desiatky dolárov.

Naviac bol vrámci práce vyvinutý front-end pre hlasovanie, ktorý umožňuje používať hlasovací systém z webového prehliadača. Ide o single-page responzívnu webovú aplikáciu vyvinutú pomocou knižníc ReactJS, React-Rexux, Material UI a Anchor web3.

# Decentralized E-Voting on Solana Blockchain

## Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Ing. Ivan Homoliak Ph.D. All the relevant text information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Martin Hošala
May 25, 2022


## Acknowledgements

First of all, I would like to thank God for making all this possible, encouraging me, and supporting me all the time. My thanks also belong to my supervisor Ing. Ivan Homoliak Ph.D, who was always happy to help me and was patiently guiding me.

# Contents

# Chapter 1

# Introduction

Voting is an integral part of democratic governance or common to a particular group. Eligible participants can vote for their preferred candidate, politician, or option by secret ballot. The voting results in the announcement of winners based on a tally of votes. Voting is usually centralized, which brings with it many negative consequences. Among them are, for example, the easy possibility of manipulation of the results, censorship, or limiting the availability of a service to a specific group of people.

The problems of centralized voting can be solved using blockchain technology, which runs a decentralized consensus protocol relying on an honest majority of all consensus nodes. Blockchain is today a rapidly growing decentralized technology with great potential that promises to improve several aspects of many industries. Among other things, blockchain technology with smart contract platforms provides exciting features that can be used to solve current e-voting problems.

Various blockchain-based e-voting systems have been proposed in recent years, primarily focusing on board voting or small-scale elections. In particular, McCorry et al. have made a significant contribution in the area of blockchain-enabled board voting by proposing a privacy-preserving self-counting voting approach called the Open Vote Network (OVN). However, OVN supports only two voting options, does not assume erroneous participants during protocol execution, and is therefore not robust to severe failures. Ivan Homoliak of BUT FIT has been involved in developing a voting protocol that addresses these two limitations of OVN. Together with three other collaborators, they presented BBB-Voting, a blockchain-based voting system that enables 1 out of $k$ voting. Each participant can vote for one of the k choices. The protocol additionally offers a mechanism to deal with erroneous participants. Similar to OVN, BBB-Voting provides maximum privacy of votes. Both OVN and BBB-Voting require a centralized authority, but its role is limited to adding participants to the voting and shifting the protocol phases. The communication between participants and the blockchain is semi-synchronous, i.e., each participant is expected to perform certain actions within a given time frame. Once all registered participants have cast their (blind) votes, the result can be tallied by anyone, and the blockchain verifies the result's correctness. Therefore, no manual decryption or unmasking of individual votes is required.

The BBB-Voting protocol has excellent features, but since it is developed for Ethereum, the real elections that would be held over it are limited by the speed of this platform. This is not good because currently Ethereum provides a throughput of approximately 15 transactions per second. With BBB-Voting, each legitimate voter must make at least two transactions on the network. Thus, if this were a large election with 1,000,000 participants, 37

hours of full network performance would be required dedicated to this election alone. Those elections would therefore have to take several days in reality. With 10,000,000 voters, it would take weeks.

This work focuses on the implementation of BBB-Voting for the Solana blockchain, which can have a throughput of 710,000 transactions per second with today's hardware. Thus, BBB-Voting on the Solana blockchain is much more promising in terms of real-world deployment for large elections than on Ethereum.

State of the art is described in three chapters. The first one is devoted to a general description of blockchain technologies, the second one describes the Solana blockchain in more detail and compares it with Ethereum. The last chapter of the state of the art section is devoted to the description of electronic voting, its features, and the BBB-Voting algorithm. The following chapter defines the requirements and the design of the solution. The subsequent chapter describes the implementation of the solution and the last two chapters summarize and conclude the results obtained.

# Chapter 2

# Blockchains

This cheaper is primarily based on the article blockchain technology overview [31] and book Mastering-bitcoin [2]. Blockchain is like a shared database. It is a decentralized and distributed system made up of a sequence of blocks. Blocks keep records of transactions created by participants. A transaction is a piece of data broadcast by a participant to the blockchain. Many transactions (typically hundreds to thousands) then form a block together, and the block is then permanently written to the blockchain. Blocks are written one after the other, each referencing its predecessor, and in this way a structured history of all transactions is created and the length of the blockchain grows continuously.

Blockchain has many uses, but the most common use case for blockchain is the public ledger (to which other functionality can be added). This ledger stores transaction records, which hold information about the transfers of crypto tokens, especially between participants in the network. Again, crypto tokens can have a variety of uses, but the most prevalent are payment between participants for things outside the network, such as bread, and payment for blockchain network services, such as storing data on the blockchain.

There is no overarching authority in the blockchain that allows or forbids transactions. There are consensus nodes (called miners). Each miner has its own copy of the entire blockchain and can add new blocks to it under predefined conditions. It then also sends these out to all the other miners to write them into their copy. So there is no such thing as a central, master copy of the blockchain. The blockchain relies on an honest majority of all consensus nodes. As long as a large part of the miners do not coordinate, the veracity of the data on the blockchain cannot be maliciously corrupted [17].

Compared to typical systems with a central authority (e.g., a bank or other company), blockchain has the following key features:

- **Decentralization.** In conventional centralized transaction systems, each transaction is processed by a central authority. This creates bottlenecks on the authority's servers, as the entire network load is directed there. Blockchain technology solves this problem because here transactions can take place directly between any two peer-to-peer (P2P) partners.

- **Persistence.** In systems with a central authority, the authority is responsible for the veracity and legitimacy of each transaction and typically has full control over their management. This leads to the problem that the authority can falsify stored data at will. Blockchain technology solves this problem by ensuring that each of the transactions propagating through the network must be validated and recorded in blocks distributed throughout the network. It is therefore almost impossible to

forge – it would have to be agreed upon by a large part of the mining nodes in the network.

- **Anonymity and self-governance.** In centralized systems, a user is typically registered and authenticated by the authority that manages his account. Typically, the authority has his personal information because it is also responsible, in a sense, for his behavior on the network. The authority can also block the user's account. In contrast, on the blockchain, the user interacts with his own generated address. This address is created and maintained by the user himself and he does not have to provide his personal data to anyone to obtain it. Also, a single user can generate and use many addresses, making it even more difficult to identify him or her. In addition, his communication within the network cannot be blocked until a large part of the network agrees.

- **Auditability and transparency.** In centralized systems, the authority holds the current state of the network and typically also the history. However, this history is typically not freely available for viewing. It can also happen that some part of the history is lost due to an error in the system. In contrast, on a blockchain, each of the transactions is time-stamped and freely accessible. Users can easily verify and track previous records by accessing any node in the distributed network. The probability of losing history on the blockchain is also very low, as all transactions are typically stored on many nodes around the world.

## 2.1 Transactions

A transaction is a message created by the user and broadcast to the blockchain network. Principally, it can contain any kind of data. However, the most widespread use case is the transfer of crypto tokens between participants. Another widespread use case for transactions is smart contracts, which are covered in detail in this thesis and are described below.

If a transaction is valid, it is added to a block and then written with the block to the blockchain forever. Simply put, to create transactions in the blockchain network, the user must have their crypto account. This is obtained by generating a private and public key pair in an agreed manner. The private key is used to sign transactions. The public key is the address of the account in the blockchain network, a kind of identification of the participant.

Signing transactions means encrypting them with a private key using the algorithm used on the blockchain. The authenticity of the signature can be verified using the public key. The result of encrypting the data using the private key and then decrypting it using the corresponding public key is the original data. In practice, it can look as follows. Consider that Martin generates a key pair $M_{pub}$ and $M_{priv}$ and Lydia generates a key pair $L_{pub}$ and $L_{priv}$. Martin wants to send some crypto tokens to Lydia. A typical transaction on the blockchain then looks like this:

1. Martin first uses algorithm A to generate a hash value H derived from the transaction data D (containing information on the transfer of tokens from $M_{pub}$ to $L_{pub}$).

2. Martin then encrypts the value of H using $M_{priv}$ to produce the encrypted value EH.

3. Martin broadcast a transaction containing the encrypted hash value EH and the data D to the blockchain. Creation of a transaction is shown in Figure 2.1.

4. The transaction is verified by decrypting the encrypted hash value of EH with $M_{pub}$ and encrypting the data D with the algorithm A. If both results are the same, it is a valid transaction. This is captured in Figure 2.2.

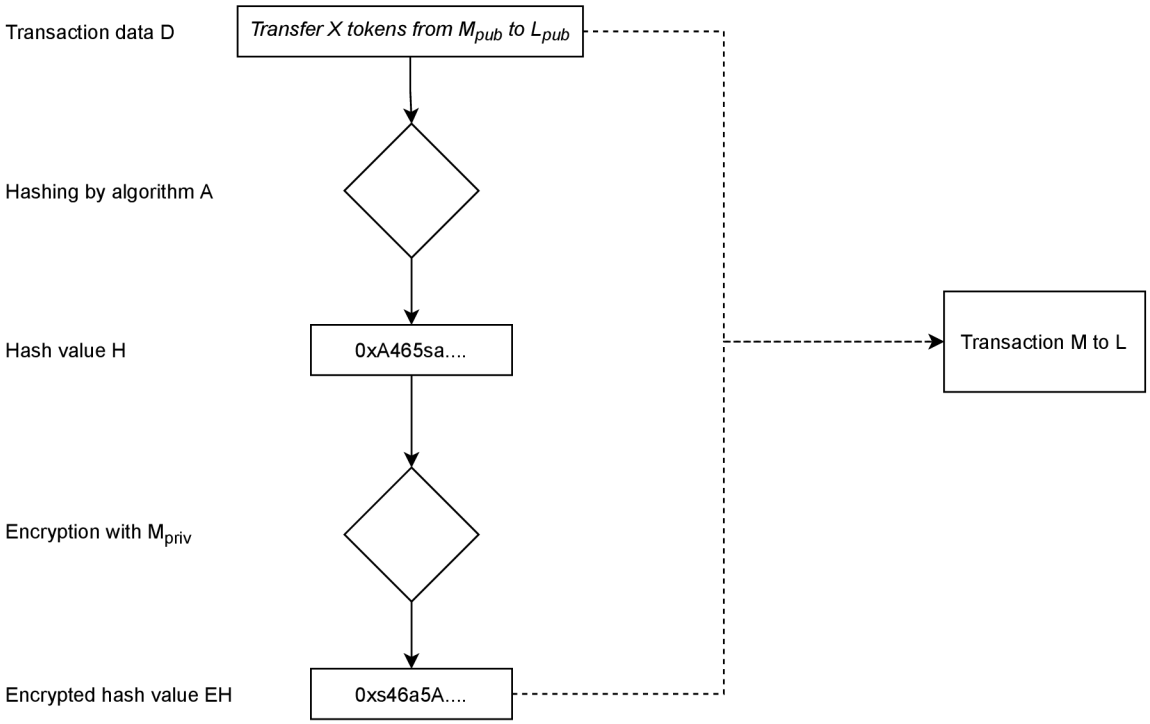


Figure 2.1: Creating transaction

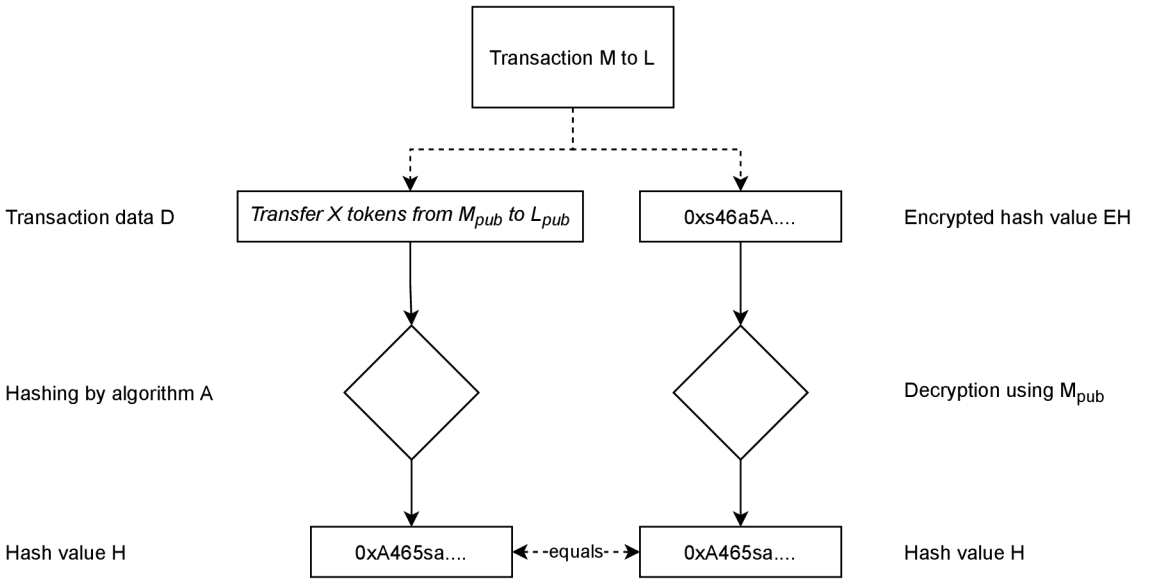


Figure 2.2: Verifying transaction.

## 2.2 Block

A block is a grouping of transactions that is time-stamped and cryptographically linked to the previous block. A block consists of a header, which contains metadata about the block, and a body, which contains a set of transactions and other related data. A block in a blockchain is captured in Figure 2.3.

Each block is added to the blockchain after being verified and passed by consensus. As new blocks are added, their cryptographic linkage makes it more difficult to modify older blocks (thus creating tamper resistance).

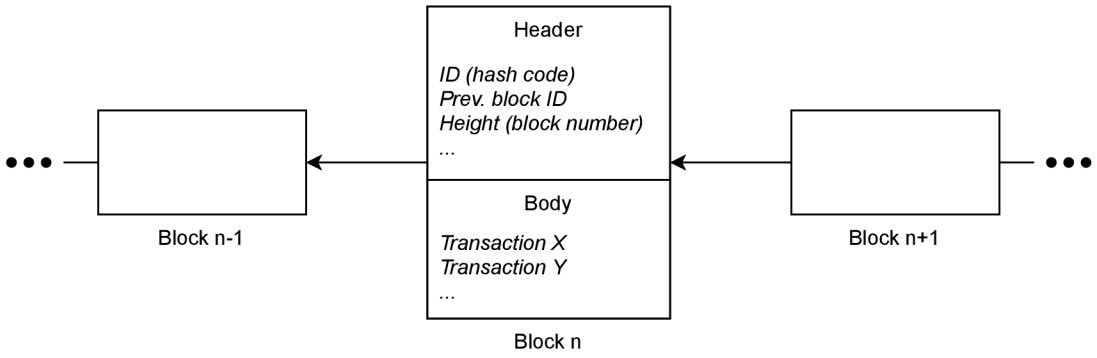


Figure 2.3: A block in a blockchain.

### 2.2.1 Consensus

Blockchain relies on the honest majority of consensus nodes. Each of these nodes has its own copy of the blockchain (not necessarily the whole blockchain) and everything needed to verify transactions and blocks, and possibly even create blocks (not every consensus node necessarily needs to be a miner as well).

Because there is no central copy of the blockchain, there needs to be an efficient, fair, reliable, and secure mechanism to ensure in real time that all transactions taking place on the network are genuine and all participants agree on a consensus state of the blockchain. There are two most common algorithms to reach consensus – Proof of Work and Proof of Stake:

- **Proof of Work (PoW)**: PoW is a piece of data requiring significant computation to find. In order to add a new block to the blockchain, a node must prove that it has done some work. Usually this involves solving some mathematical puzzle, such as finding a number called a nonce. A nonce is a number for which a prescribed function gives a result in some agreed range. This function has two inputs: the nonce and the data portion of the corresponding block. The range of the result to hit varies according to the sum of the computational power in the network – the more power in the network, the narrower the range is, to make the block harder to load. One consensus rule is that the longest valid blockchain is the true one. The length is the number of blocks, and this is specified in the header of each block. It is actually the order of the block. If a node receives a message with a block that has this number greater than the highest block number in that node, it will modify its version of the blockchain to contain this newly received block. This typically consists of simply appending the block to the

end of the blockchain. The main idea of the PoW consensus algorithm is shown in Figure 2.4.

However, it may happen that multiple miners mine a block at approximately the same point in time and thus with an equal block number. Each will start spreading its new block over the network and the other nodes will accept them. When a new block from the first miner arrives to someone who already received a new block from the second miner, they will discard it, as the block number is not higher. This creates two (or more) groups of nodes with a different blockchain.

These groups then each work on the top of their version. Whichever group mines the next block first, it is spread throughout the network. In this case, nodes from other groups will have to overwrite the previous block as well – the blocks are cryptographically linked as stated above. Theoretically, it is possible that in such a partitioned network, it is again possible that multiple blocks are mined at the same time and the whole network is not immediately merged by the next block. Nevertheless, the probability is very low and a block above which there are more than 5 others is considered to be certainly unchangable.

An important aspect of this model is that the work already invested in solving a problem does not affect the probability of finding a solution to that problem, nor does it affect the probability of finding a solution to the next problem. Thus, if someone finds a solution and publishes a given block, everyone can discard their current work and start working on a new block on top of the latest block. This implies that an underperforming node is unlikely to mine anything even in a very long time. The miners therefore cluster together and form so-called mining pools. If someone in the group manages to find a new block, the reward is divided fairly among all members.

However, forming such pools goes against the principle of decentralization. When a group gains a supermajority of the network's power, it can ignore blocks from others and control the whole network. On the other hand, it turns out that it is in the interest of the miners to make the network work in the first place and for people to have confidence in it. In 2014, the Ghash.io pool probably exceeded 50 % of the network's computing power. Thus, it could theoretically ignore the work of everyone else, mine all the blocks itself, and get significantly higher rewards, possibly even validate fake transactions. Yet, no such thing occurred. The pool itself has issued a statement that it in no way intends to harm-bitcoin and that it will not exceed 40 % in the future. Pool miners began to voluntarily move to other pools. At the time of writing this essay, no pool exceeds 20 % of the network's mining capacity.

PoW principle is used by the most popular cryptocurrency networks such as-bitcoin and Ethereum. In the-bitcoin network, to mine a block, the miner must find a numerical solution to the SHA256 algorithm, and the difficulty is regulated so that one block is mined in an average of 10 minutes. Ethereum's block time is about 15 seconds and Ethereum miners must find a numerical solution to the Ethash algorithm.

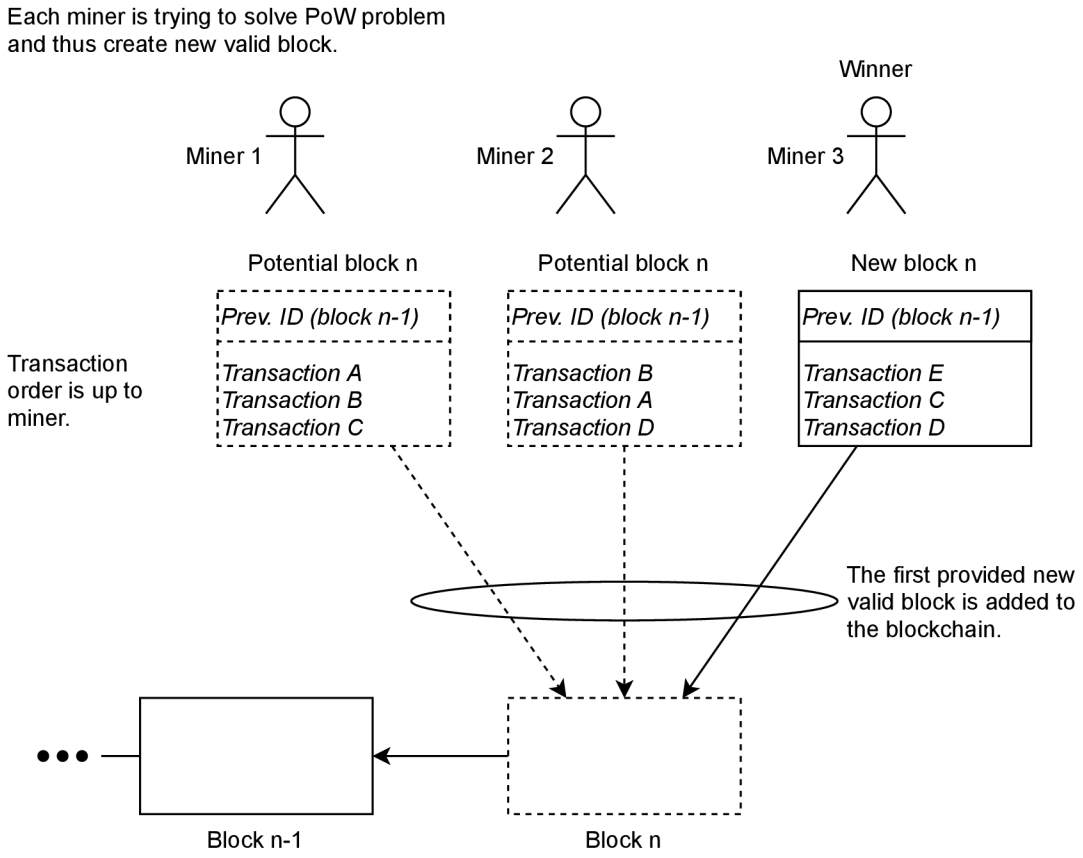


Figure 2.4: Principle of the Proof of Work consensus algorithm.

- **Proof of Stake (PoS)**: A cheap and energy-efficient alternative to the PoW algorithm is the consensus algorithm PoS. There is no need to perform any power-intensive calculations in PoS. Each miner has to lock some capital, most often crypto tokens of a given network, by a special transaction. The tokens are thus staked and cannot be used during the time they are staked. For each new block, a node is selected to verify the block and receive a reward for it. This selection can happen in several ways, but the principle is always that the larger the stake, the more likely a given node will be selected. The main idea of the PoW consensus algorithm is shown in Figure 2.5.

  This system relies on the fact that the greater the user's stake, the less incentive he should have to commit malicious activity on the network. When malicious activity happens in the network, the value of the network tokens can drop significantly, which would also have a negative impact on the locked tokens of the malicious user. In addition, established blockchain networks would be very costly to take control over. Thus, this system can be used if enough people put resources into the blockchain network. Initially, a different method must be used to reach consensus and once the blockchain community is firmly established, it is possible to move to a proof of stake.

  The selection of a block issuer can either be random, where the blockchain network looks at all users with a stake and chooses from among them based on the ratio of

their stakes to the total amount of cryptocurrency staked. So if a user has staked 1% of all staked tokens, he is on average selected once in a hundred blocks.

Another way of selecting a block issuer is by a multi-round voting system. The blockchain network selects several users with a stake. Each of the selected users creates a block and then all users with stakes vote for one of these blocks. Several rounds of voting may take place before a new block is decided. This method allows all users with a stake to participate in the creation of each new block.

The selection of the issuer of a block can also be done through a coin age system called coin age proof of stake. In this case, the staked cryptocurrency has the property of age. After a certain amount of time (e.g., 30 days) has elapsed, the staked crypto token can be factored into the probability of its owner's selection to publish the next block. If the user is not selected, as time passes, the non-winning tokens gain weight and thus increase the probability of their holder winning. When their holder wins, the age of the tokens is reset and they must again wait a specified period of time.

Another way in which block issuer selection can take place is through the delegate system. Users vote for nodes to become publisher nodes – that is, they create blocks on their behalf. The voting power of blockchain network users is tied to their share, so the larger the share, the more weight the vote carries. The nodes that receive the most votes become publishing nodes and can validate and publish blocks. Blockchain network users can also vote against an established publishing node and try to remove it from the set of publishing nodes.

Of the well-known blockchains, Cardano, for example, uses PoS, but the second largest cryptocurrency Ethereum is also planning a change to PoS in 2022. Future versions of Ethereum could also switch to a delegative PoS system.

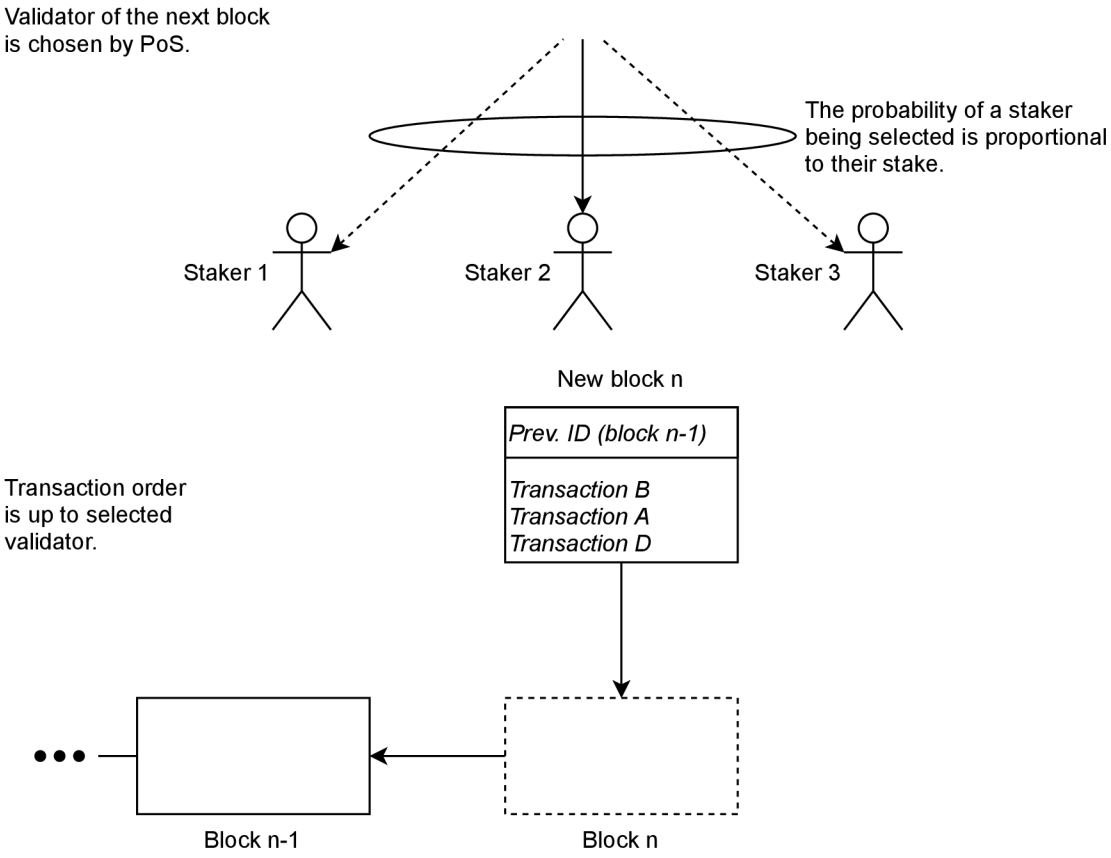


Figure 2.5: Principle of the Proof of Stake consensus algorithm.

# Chapter 3

# Solana

Solana [32] is a new, high-performance, permission-less blockchain. The project is open source and it is maintained by Solana Foundation based in Geneva, Switzerland. Solana proves that the theoretical limit for a centralized database – to process 710,000 transactions per second on a standard gigabit network if transactions are no larger than 176 bytes on average – apply equally well to a blockchain on an adversarial network. This is possible when individual nodes can rely on time and Solana creators developed a way to share reliable time without nodes having to rely on one-another. They called this mechanism Proof of history and it is described in the section below.

Solana's native token is SOL. SOL can be used natively as payment for running a program on the blockchain or verifying its output. Solana's system makes micro-payments in sub-SOLs called lamports. A lamport is the smallest, natively indivisible unit within the network, has a value of 0.000000001 SOL, and is named after Leslie Lamport.

## 3.1 Proof of history

Proof of history (PoH) [32, 31] is another approach to reaching consensus. It is a subtype of proof of stake, but it provides a revolutionary way of synchronizing individual nodes in the network. Individual nodes in the network process transactions and mark them with objective timestamps that nodes in the network can rely on, allowing them to optimistically trust the order and timing of messages before reaching consensus. Consensus then comes later. Participants within the network periodically vote on what they consider to be the main branch of the blockchain. Each time they do so, they commit not to give their vote to another branch for a certain period of time. Their commitment not to vote for another branch then grows exponentially the more they vote for one particular branch. To actually vote for what they believe to be the true blockchain they are motivated by the fact that until they accumulate 32 votes for one particular branch, they will receive no rewards.

## 3.2 Entries in Solana blockchain

An entry in the Solana blockchain is called an account. An account contains its metadata and either contains some other data or is an executable program – then it is called a program. In addition, a Solana account can contain funds similar to an account in a traditional bank. These are called lamports. A Solana account is addressable by a key, referred to as a public key (pubkey); if it is a program, its address is called the id.

Each Solana account has an owner. This owner must be a program, and the account has its id stored in the metadata. The created account is initialized to be owned by an embedded program called System Program and is called the system account. The program has write access to the account if its id matches the owner. If the owner is System clients who present the private key for the account can transfer the lamports and also change the owner to a different program id. A program that does not own the account can only read its data and credit lamports.

Accounts are marked as executable during a successful deployment process by the loader that owns the account. When a program is deployed to the execution mechanism (BPF deployment), it is determined by the loader whether the byte code in the account data is valid. If so, then the program account is permanently marked as executable and owned by BFP Loader. Deployed programs can be executed by any Solana account.

## 3.3 Executing programs

The client application communicates with the Solana cluster using a transaction with one or more instructions. Solana passes these instructions to accounts that are already deployed as programs on the blockchain. For example, an instruction might instruct a program to transfer lamports from one account to another, or create an interactive contract that controls how lamports are transferred. Transaction instructions are executed sequentially and atomically – if any of them fail all changes made by the transaction are undone.

An instruction consists of a single program id, a subset of transaction accounts, and a data byte array. The program id in an instruction specifies the program that will perform the processing of that instruction. The program account owner determines which loader to use to load and execute the program. Program execution by instruction is displayed in Figure 3.1.

The accounts referenced by the instruction constitute on-chain state and are used both as inputs and outputs of the program. If the program needs to store a state between transactions, it will use accounts to do that. Accounts can serve a program similarly to files in operating systems such as Linux because they can persistently store arbitrary data. Also, like a file, an account contains metadata, with information about who can access the data and how. However, the accounts are stored in the validator's memory and pay rent to stay there. Validators periodically check all accounts and collect the rent. If an account runs out of lamports it is deleted. Accounts can be marked as rent exempt if they contain a sufficient number of lamports – currently this is a two year rent.

The byte array that the instruction carries is passed to the program. The data content of an instruction is program-specific and usually used to specify the operations to be performed by the program as well as any additional information that those operations may need in addition to what is contained in the accounts. Programs can freely specify how information is encoded into the instruction data array. However, this choice should take into account the overhead of decoding, since this step is performed by the program on-chain.

### 3.3.1 Sealevel

Solana uses a parallel smart contracts run-time named Sealevel. Solana is able to process transactions in parallel because Solana transactions describe all the states – referencing all the accounts – that a transaction will read or write during execution. This makes it possible

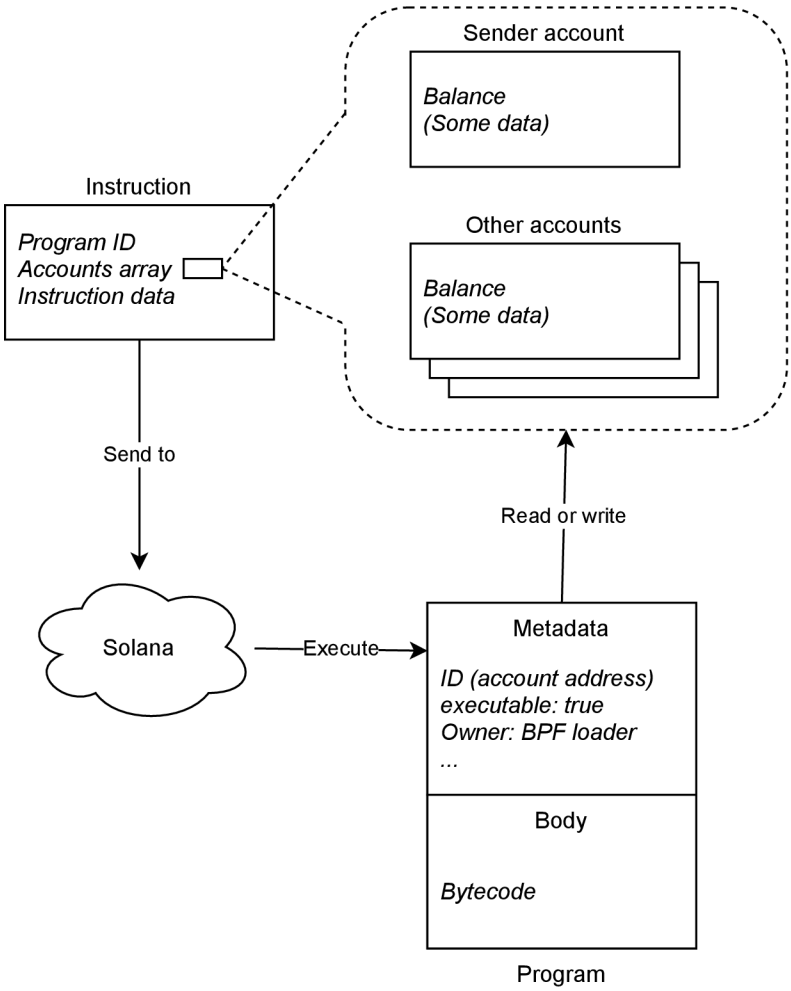


Figure 3.1: Solana program execution by instruction.

to execute non-overlapping transactions concurrently, and also to execute transactions that only read the same account concurrently.

Furthermore, since a solana program has no state but is just executable bytecode, Sealevel can execute a single program concurrently over a different vector of accounts. Modern GPUs have around 50 multiprocessors that can execute the same code over around 80 different inputs. This means that programs only need to be deployed once, and different users can use them very efficiently concurrently as long as they don't need to write to the same account.

## 3.4 Comparison to Ethereum

Ethereum [30] is the first blockchain platform capable of executing smart contracts. It has a very developed community and the trust of many people. Compared to Solana, however, it is much slower, only able to perform about 15 transactions per second, whereas Solana can perform hundreds of thousands.

To achieve consensus, Ethereum uses the Proof of Work algorithm, which is one of the main reasons for the low speed. However, at the time of writing, Ethereum has begun the process of upgrading to Ethereum 2.0, which will use the Proof of Stake protocol and promises to increase speed to tens of thousands of transactions per second [5].

### 3.4.1 Accounts

An entry in the Ethereum blockchain is also an account. Similar to Solana, it is an entity with a balance – in this case, ether (ETH). Also, there are two types of accounts, user-managed (externally-owned) or accounts deployed as smart contracts (contracts).

The main difference from Solana is the fact that externally-owned accounts used in Ethereum do not carry any data. They only serve to store balances and create transactions. On the other hand, Ethereum Smart contract is not stateless like the program in Solana. It has a data store where it can store arbitrary data [18].

### 3.4.2 Execution of smart contacts

In Ethereum, as in Solana, they trigger the execution of some code on the blockchain of a transaction from an externally-owned account to a smart contaract account. This code can then perform many actions, such as transferring tokens or even creating a new contract.

The biggest difference is that Solana offers parallel running of smart contracts, whereas in Ethereum only one smart contract runs at a time. The others are waiting in a queue. This is because Ethereum uses EVM which is a special state machine keeping all Ethereum accounts and smart contracts live. The EVM exists as a single entity which is maintained by thousands of computers running the Ethereum client. Every smart contract in Ethereum changes the state of this machine and this state must be synchronized across the network [6].

# Chapter 4

# E-Voting

Voting is an essential part of the governance of public affairs, or the governance of things common to a group. This can be a vote within a corporate board of directors or a vote in a national election. E-Voting protocols are protocols that allow stakeholders to participate in voting in the broadest sense using the computer technology.

## 4.1 Properties of e-voting protocols

E-voting protocols are expected to satisfy certain properties. These properties are important to ensure that the protocols are reliable and bring benefits to the participants, so that switching to it will not bring problems, but benefits compared to the traditional way of voting. A list of such properties appears in [23, 7, 14]:

**Privacy of the Vote:** Ensures the secrecy of the contents of the ballot paper. In a vote that satisfies this property, a participant's vote may not be revealed except by the participant himself (by his own decision) or by collusion of all other participants. In large-scale elections, privacy can largely be achieved by trusted authorities. For small-scale voting, it can be achieved by interaction between voters or by homomorphic encryption. However, vote privacy can also be violated in protocols using homomorphic encryption, although all ballots remain secret. This can happen in the case of a unanimous vote of all participants. This property is acceptable if and only if the voting protocol is required to output the final vote total as a result (which is also the case for BBB-Voting). However, if the voting protocol is required to output only the winning choice or the order of the candidates, then the voting privacy property is satisfied only if the information about the unanimous vote is not extractable from the protocol output.

**Perfect Ballot Secrecy:** Is an extension of vote privacy. It means that a partial sum (finding partial results before the vote is closed) is only available if all remaining participants participate in the vote recovery.

**Fairness:** The total can only be calculated after all participants have cast their votes. Therefore, no partial totals can be revealed to anyone before the voting log is closed.

**Verifiability:** Each participating participant can verify that all votes cast are legitimate and the final total is correct. This is achieved by using a tamper-resistant public bulletin board (in the case of BBB-Voting, this is the blockchain) and zero-knowledge

proofs to verify the validity of the votes cast. Voting systems that satisfy this property are also referred to as end-to-end verifiable voting systems.

**Self-Tallying:** After all votes have been cast, any party may perform a recount. However, systems providing this property have to deal with fairness problems, since the last participant can theoretically compute the total before casting his vote. This can be remedied with an additional verifiable dummy vote. A useful feature of self-counting protocols is that it is not necessary for votes to be manually uncovered in order to determine the results of the vote.

**Dispute-Freeness:** The voting protocol contains built-in mechanisms to eliminate disputes between participants; therefore, any third party is able to verify that an active participant has followed the protocol. The protocol should have a publicly verifiable audit trail, which contributes to the reliability and trustworthiness of the scheme.

**Fault Tolerance:** The voting protocol is able to recover from erroneous participants, with errors being publicly visible and verifiable due to nonrepudiation. This is possible if the remaining honest users who correct the errors participate in the recovery. In this way, the error-resilient protocol „removes“ participants who left during the voting process.

**Resistance to Serious Failures:** Serious failures refer to situations in which the results of a vote have been altered (either by simple error or by a hostile attack), and such an alteration may be unrecoverable and, if detected, is unrecoverable without restarting the entire vote.

## 4.2 BBB-Voting

This section is based on BBB-Voting paper [29]. BBB-Voting meets all the requirements for e-voting protocols described in the previous section. It provides perfect ballot secrecy, fairness, public verifiability, self-tallying functionality, dispute-freeness, resistance to serious failures, and maximizes voter privacy. Due to the fact that the BBB-Voting protocol satisfies each of the above desired properties named in Section 4.1 while offering the possibility to vote from k candidates, it can be considered as one of the best voting protocols of the present time. A comparison of this protocol with other concurrent voting protocols can be found in Table 4.1.

This protocol publishes a full tally at the output and uses homomorphic encryption to achieve vote privacy and perfect ballot secrecy. In addition, it supports a fault tolerance mechanism that allows the protocol to recover from a fixed number of participants who have not cast their votes without restarting. Fault tolerance support also increases the protocol's resistance to serious failures. To achieve fairness, the authority casts a public dummy vote after the deadline, which marks the end of the voting.

### 4.2.1 Phases

BBB-Voting protocol has five phases which are captured in Figure 4.1. Voting with the BBB-Voting protocol the goes as follows:

1. **Registration:** The Voting Authority (VA) first verifies each participant's proof of identity. This can either be verifiable credentials signed by the issuer in decentral-

| Approach | Privacy of Votes | Perfect Ballot Secrecy | Fairness | Self-Tallying | Fault Tolerance | Uses Blockchain | Verifiability | Src. Code Available | Choices |
|---|---|---|---|---|---|---|---|---|---|
| Hao et al. [8] | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | 2 |
| McCorry et. [22] | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | 2 |
| Seifelnasr et al. [25] | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | 2 |
| Li et al. [20] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | 2 |
| Matile et al. [21] | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | $k$ |
| Kiayias and Yung [14] | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | $2/k$ |
| Khader et. [13] | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | 2 |
| Zhang et al. [33] | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | $k$ |
| Baudron et al. [3] | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | $k$ |
| Groth [7] | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | $k$ |
| Adida [1] | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ | $k$ |
| Killer [15] | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | 2 |
| Shahandashti and Hao [27] | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | $k$ |
| Rui and Ribeiro [12] | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | $k$ |
| Sandler et al. [24] | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ | $k$ |
| Bell et al. [4] | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ | $k$ |
| **BBB-Voting** [29] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | $k$ |

Table 4.1: A comparison of features and properties of various voting protocols. [29]

ized identity management, or proof of identity interactively provided by a third-party identity provider in centralized identity management. Nevertheless, this identity verification is not an integral part of the protocol and it is up to the VA to decide how to achieve it.

The VA then collects the blockchain wallet address, i.e., the public key (PK), from all authenticated participants. Once the VA has the wallet addresses, it creates a voting smart contract (VSC) and registers all verified participants into the voting. This registration is done within a transaction that deploys VSC to the blockchain by submiting all the wallet addresses of voters eligible for the particular voting. The deployment transaction also includes maximum time limits for all subsequent phases of the protocol.

2. **Setup:** At this stage, the participants agree on the parameters of the vote. These parameters are set up by VA as a part of the VSC deployment transaction. Voting parameters are universal, publicly visible and can be accessed at VSC address. Any participant and the public can thus verify these parameters.

   If the participant is satisfied with the parameters, it proceeds further by generating its ephemeral private key and ephemeral public key. He then sends his ephemeral public key to the VSC in a transaction signed by the wallet whose address he provided to the authority in the first phase. This step commits to sending the vote at a later time. As part of this transaction, the participant also sends a deposit, which he will get back after the vote is completed. However, should the participant fail to vote within the time limit (or fail to participate in any failure recovery), his deposit will be distributed to the other participants.

If a participant does not find the parameters sufficiently secure, or disagrees with them for any reason so they do not intend to vote, they may not submit their public ephemeral key. Voting then continues without them while they are not subject to any penalty.

3. **Pre-Voting:** In this phase, the handles are synchronized between all participants to achieve the self-tallying property. The synchronization is performed by a multi-party computation (MPC), which however does not require any interaction between the participants since all ephemeral public keys are already published in the SC. Thus, the MPC keys can be computed by the VSC itself and this computation is triggered by the authority sending the transaction. The VSC computes and stores the MPC key for each participant.

   Each participant can retrieve the MPCs of all participants. Using their private ephemeral key, each participant can thus calculate their own ephemeral blanking key.

4. **Voting:** At this stage, each participant wraps, blinds and casts their vote to the VSC. Blanking means that the participant multiplies their vote choice by their blanking ephemeral key. The participant sends such a homomorphically encrypted vote choice to the VSC along with a non-interactive proof of 1-of-k set membership (NIZK). The VSC verifies the correctness of the NIZK proof to ensure that the blinded vote is well-formed, and then stores the blinded vote.

5. **Tally:** In this last stage, it is necessary to guess by brute force the vector of vote totals obtained for each of the options. Guessing can be done, for example, using exhaustive search, where the maximum number of attempts for guessing is n+k-1 over k. The guessing is done by having either party offchain construct a vector representing the possible vote outcome and send it to the VSC. The VSC then verifies whether the result is correct and, if so, terminates the voting and announces the result to all parties. Although the exhaustive search for a 1 out of $k$ vote is computationally intensive as opposed to voting, the process can be parallelized significantly. This is because it is not necessary to send each vector representing a possible voting result to the VSC and wait for verification. This verification can also be done offchain using data publicly available on the VSC.

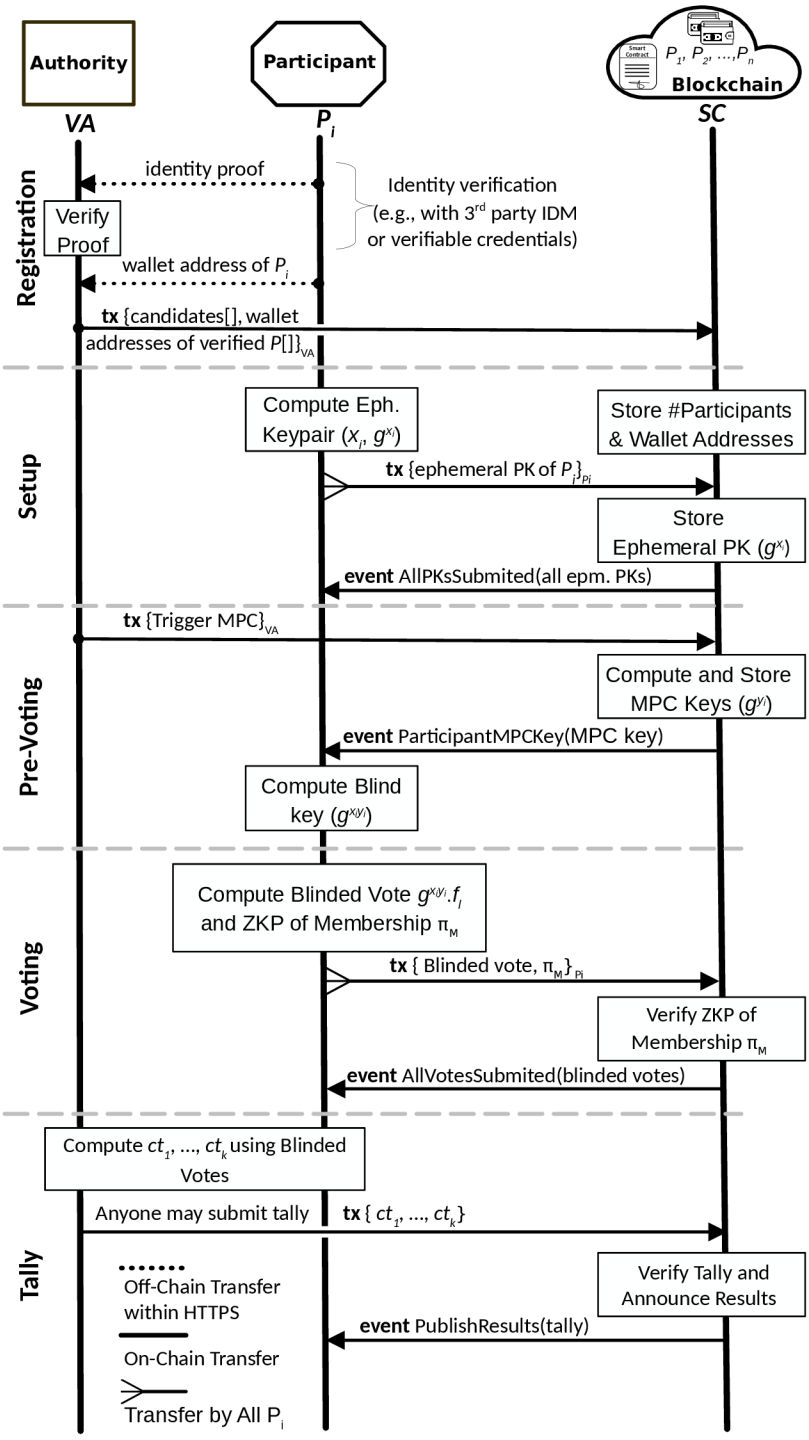


Figure 4.1: Phases of the BBB-Voting protocol [29].

## 4.3 Scalable extension

Ivana Stančíková from BUT FIT created a scalable extension of BBB-Voting as her master thesis [28]. Principally, this extension works in such a way that voters are divided into groups and voting contains a main contract `MainVotingC` and sub-contracts for groups of

voters `VotingBoothC`. For the creation of `VotingBoothC` contracts, the main voting contract is not used directly, but they are created using the auxiliary `VotingBoothDeployer` contract. Voters vote within the `VotingBoothC` contracts and the results are later aggregated back into the main contract. The interaction and structure of these contracts is shown in Figure 4.2.

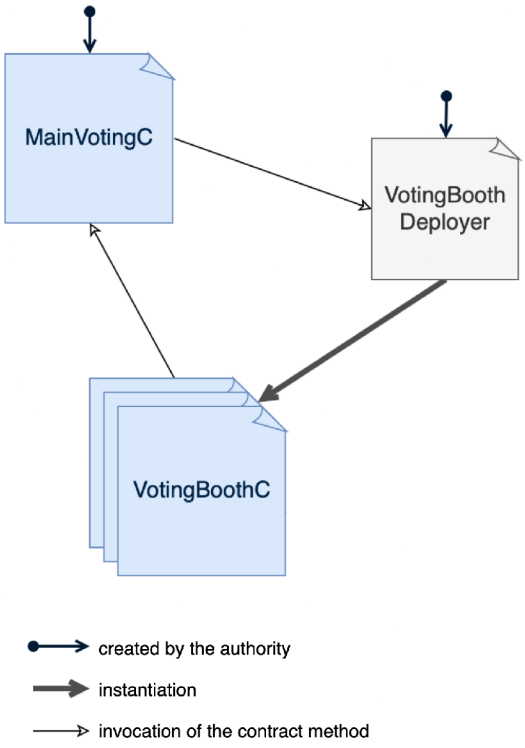


Figure 4.2: Illustration of voting system structure with scalable extension and interaction between contracts. [28]

A prerequisite for starting voting is the deployment of all necessary smart contracts. This step is performed by the authority administering the voting. The MainVotingC main contract constructor is already passed the candidate data, i.e. their description and cryptographic generators, when it is created. If these contracts are ready on the blockchain, the individual steps of the voting protocol can proceed. The voting protocol is divided into six phases identical to the BBB-Voting protocol with an extension for vote repair - compared to the phases described in the previous section, there is still a Fault Repair phase between the Vote and Tally phases. The `MainVotingC` contract and the individual `VotingBoothC` instances maintain information about the current phase, and when functions of these contracts are called (either by the authority or by the voter), it is checked whether the voting is in the phase for which the function is relevant.

In the Setup phase, after passing the addresses of all eligible voters to the main contract, the authority starts dividing the voters into groups. The splitting is performed by `MainVotingC` and may again be done sequentially in multiple transactions. The required number of groups is defined by the authority. The contract assigns a group index to each of the voter addresses it received earlier and stores this assignment in its storage. Separate contracts are created for the voting groups that resulted from the previous voter distribu-

tion. The splitting of the creation of these contracts into smaller transactions is handled by the authority as in the previous cases. In the repository of each child contract, the data needed for voting - the number of voters belonging to a given group, candidate data, and cryptographic parameters - are defined when the contract is created.

In the Signup phase, the `VotingBoothC` child contract is ready to register the voters' public keys. Voters first obtain information about which group they have been assigned to and the address of the corresponding child contract from the master contract repository. After that, voters only interact with their assigned `VotingBoothC` contract. At this stage, they register their public keys in the contract. When the public keys of all voters of a given group are registered in the contract, the contract moves to the next phase.

In the Pre Voting phase, the calculation of the public parts of the voting keys of all voters in each of the subsidiary contracts. The computation in `VotingBoothC` contracts is triggered by the authority. A voter can retrieve the public keys of the other voters from the contract and compute the public part of the voting key locally, or can retrieve the resulting key directly from the contract after this phase.

In the Vote phase, the voter sends his blind vote and the corresponding NIZK proof to the `VotingBoothC` contract of the corresponding group. The contract performs a verification of the attached proof. If the verification is successful, the contract saves the voter's vote. If all voters who have previously provided their public keys have cast their ballots, the contract moves to the Tally phase and notifies the `MainVotingC` contract. However, if any of the voters are inactive at this stage (do not cast a ballot), one of the participants must trigger the start of the Fault Repair phase.

When moving to the Fault Repair phase, voters who did not participate in the ballot casting are found. The other voters are informed about them. Each active voter must create correction keys (one for each inactive voter) for their ballot as well as proofs of correctness of these keys. This information is forwarded to the appropriate `VotingBoothC` contract, and the `VotingBoothC` will both verify the evidence and correct the ballot it received from the voter in the Voting phase. Once the ballots of all active voters have been corrected, the contract moves to the Tally phase and notifies the main contract at the same time.

The first step of the Tally phase is to calculate the results in each subsidiary contract. This calculation is not performed directly by the individual contracts, but by the authority. The result is then sent by the authority to the contracts for validation. If the vote result is correct, it is sent to the master contract for aggregation. The `MainVotingC` contract moves to the Tally phase when it receives confirmation of moving to this phase from all child contracts. It then receives the results from each group and aggregates them into the final result. This is only available to the voting participants after the results of all child contracts have been delivered to the main contract.

## 4.4 BBB-Voting implementation

There are 2 different ways to implement BBB-voting. The first, more straightforward one is to implement integer arithmetic. This method can be considered secure when using ephemeral keys and multiparty keys of size at least 1024-bits and is described in the BBB-Voting paper. The second method is an implementation using elliptic curve cryptography.

For solving this work, two implementations of the protocol have been provided. Both solutions are designed for Ethereum and implemented in the Solidity language. The first one is a solution using integer arithmetic, but using 128-bit keys. Solidity natively supports a maximum of 256-bit numbers, and since multiplying a number doubles the number of-bits

to store the result and key multiplication occurs within the protocol, a maximum of 128-bit keys can be used natively on Solidity. So this implementation is not safe, but it is sufficient as a proof-of-concept, since extending it to 1024-bit keys is possible – it would just be more complex and complicate the code, since the native Solidity language elements could not be used.

The second solution uses elliptic curve cryptography and is thus secure. However, it is not so straightforward – the smart contract code is almost 3 times longer and besides, this solution also contains its own libraries implementing elliptic curve operations.

Both solutions are implemented in Solidity as one smart contract named BCVoting. This contains in both cases methods reflecting the phases of the BBB-Voting algorithm:

1. **Registration:** In the first phase, the authority invokes the `constructor` to create the contract. It then adds the participants to the VSC using the method `enrollVoters`
2. **Setup:** At this stage, the participants confirm their interest to vote by sending their ephemeral public key to the VSC using `submitEphemeralPK` method.
3. **Pre-Voting:** Although each participant can compute its own multiparty key from the ephemeral keys of all participants available in the VSC, this computation will also take place in the VSC itself. This is because VSC will need the keys to validate the votes received in the next phase. The MPC computation within the VCS is triggered by the authority using the `computeMPCKeys` method.
4. **Voting:** Participant wraps, blinds and send their vote along with NIZK using the VSC method `submitVote`.
5. **Tally:** Guessing and verifying of the vector of votes each candidate obtained can be done off-chain. However, once the result is guessed, it can be sent to the VSC using the method `computeTally`. This method verifies the result and if it is correct stores it in the VSC where it remains publicly available.

In the implementation using elliptic curve cryptography, the computation of multiparty keys in the pre-voting phase is divided into two parts. In the first part of the computation the so-called right markers are computed and in the second part the key computation is completed. This approach has two advantages. The first one is that the computational effort is split into two transactions and thus the maximum allowed computational load – the gas limit on Ethereum – is not exceeded. The second one is that right markers do not have to be recalculated when recovering from a VSC faulty state, i.e. from a situation where one of the participants fails to send their vote and new multiparty keys have to be calculated in order to resume voting. A more detailed description of the computations in both kinds of implementations can be found in the following section.

### 4.4.1 Equations and calculations used within the algorithm

This section describes the calculations that take place in each stage of the algorithm. For each phase, the calculation used in the integer arithmetic version of the voting algorithm is described first. This is followed by a description of the version using elliptic curve cryptography.

1. **Registration:** First it is necessary to select a common generator $g \in \mathbb{F}_p^*$. Authority choses a safe prime $p$, i.e., $p = 2 \cdot q + 1$, where $q$ is a prime. This is to ensure that

the multiplicative group of order $p-1=2\cdot q$ does not have any non-trivial to detect small subgroups. Let $n$ be the total number of participants such that $n<p-1$. Any participant $P_i$ can later submit a vote $\{v_i \mid i \in \{1,2,...,k\}\}$ for one of $k$ choices by selecting $k$ independent generators $\{f_1,...,f_k\}$ in $\mathbb{F}_p^*$.

These generators should meet following property to prevent having more than one different valid tallies:

$$f_i = \begin{cases} g^{2^0} & \text{for choice 1,} \\ g^{2^m} & \text{for choice 2,} \\ \quad \ldots & \\ g^{2^{(k-1)m}} & \text{for choice k,} \end{cases} \tag{4.1}$$

where $m$ is the smallest integer such that $2^m > n$ (the number of participants).

2. **Setup:** In this phase, each participant $P_i$ generates his ephemeral private key as a random number $x_i \in_R \mathbb{F}_p^*$ and computes ephemeral public key (EPK) as $g^{x_i}$. EPK is then sent to the VSC.

   For ECC, the following applies: ephemeral private key $x_i \in_R \mathbb{Z}_{nn}$ and EPK is $x_i \cdot G$.

3. **Pre-Voting:** Now the authority triggers the computation of MPC keys which causes the VSC to compute and store the MPC key for each participant as follows:

$$h = g^{y_i} = \prod_{j=1}^{i-1} g^{x_j} / \prod_{j=i+1}^{n} g^{x_j}, \tag{4.2}$$

   where $y_i = \sum_{j<i} x_j - \sum_{j>i} x_j$ and $\sum_i x_i y_i = 0$. As the $y_i$ is derived from the private ephemeral keys of other participants, it is secret. Therefore despite the fact that anyone can compute $g^{y_i}$, $y_i$ can only be revealed if all other participants than $P_i$ cooperate or by solving the discrete logarithm problem (DLP) for Equation 4.2. Since each $P_i$ can obtain $g^{y_i}$ of all participants, he can compute $g^{x_i y_i}$ (ephemeral blinding key) using his private key $x_i$.

   If ECC is used, MPCs are computed as follows:

$$H_i = y_i \cdot G = \sum_{j=1}^{i-1} x_j \cdot G - \sum_{j=i+1}^{n} x_j \cdot G \tag{4.3}$$

   The ephemeral blinding key of participant $P_i$ is computed as $x_i \cdot H_i$, where $H_i$ is found in Equation 4.3 by the self-tallying property, $\sum_{i=1}^{n} x_i \cdot H_i = 0$.

4. **Voting:** Here, each participant $P_i$ needs to blind and submit his vote to $VSC$ along with a 1-out-of-$k$ non-interactive zero-knowledge (NIZK) proof of set membership to $VSC$. The $VSC$ must verify the correctness of the blinded vote. Participant $P_i$ blind his vote as follows:

$$B_i = \begin{cases} g^{x_i y_i} f_1 & \text{if } P_i \text{ votes for choice 1,} \\ g^{x_i y_i} f_2 & \text{if } P_i \text{ votes for choice 2,} \\ \quad \ldots & \\ g^{x_i y_i} f_k & \text{if } P_i \text{ votes for choice } k. \end{cases} \tag{4.4}$$

Method by which the contract verifies the set membership is shown in Figure 4.3. When the contract verifies the validity of the proof, it stores the blinded vote.

If ECC is used, blinded vote is calculated as follows: Let $\{F_1, F_2, ..., F_k\}$ be $k$ independent generator points on the curve that generates the same subgroup as $G$. The blinded vote options for a participant $P_i$ are

$$V_i = \begin{cases} x_i y_i G + F_1 & \text{if } P_i \text{ votes for candidate 1} \\ x_i y_i G + F_2 & \text{if } P_i \text{ votes for candidate 2} \\ ... & \\ x_i y_i G + F_k & \text{if } P_i \text{ votes for candidate } k \end{cases} \tag{4.5}$$

The VSC voice verification in this case is shown in the figure 4.4 and 4.5.

5. **Tally:** When the voting ends the results are calculated off the chain and when a correct result is found it is sent to $VSC$. The result is represented as the number of votes $ct_i, \forall i \in \{1, ..., k\}$ of each candidate. Calculating the results means finding the numbers $ct_i$ that satisfy the equation 4.6.

$$\prod_{i=1}^{n} B_i = \prod_{i=1}^{n} g^{x_i y_i} f = g^{\sum_i x_i y_i} f = {f_1}^{ct_1} {f_2}^{ct_2} ... {f_k}^{ct_k}. \tag{4.6}$$

This can be achieved by an exhaustive search which is bounded above by a number of attempts equal to $\binom{n+k-1}{k}$.

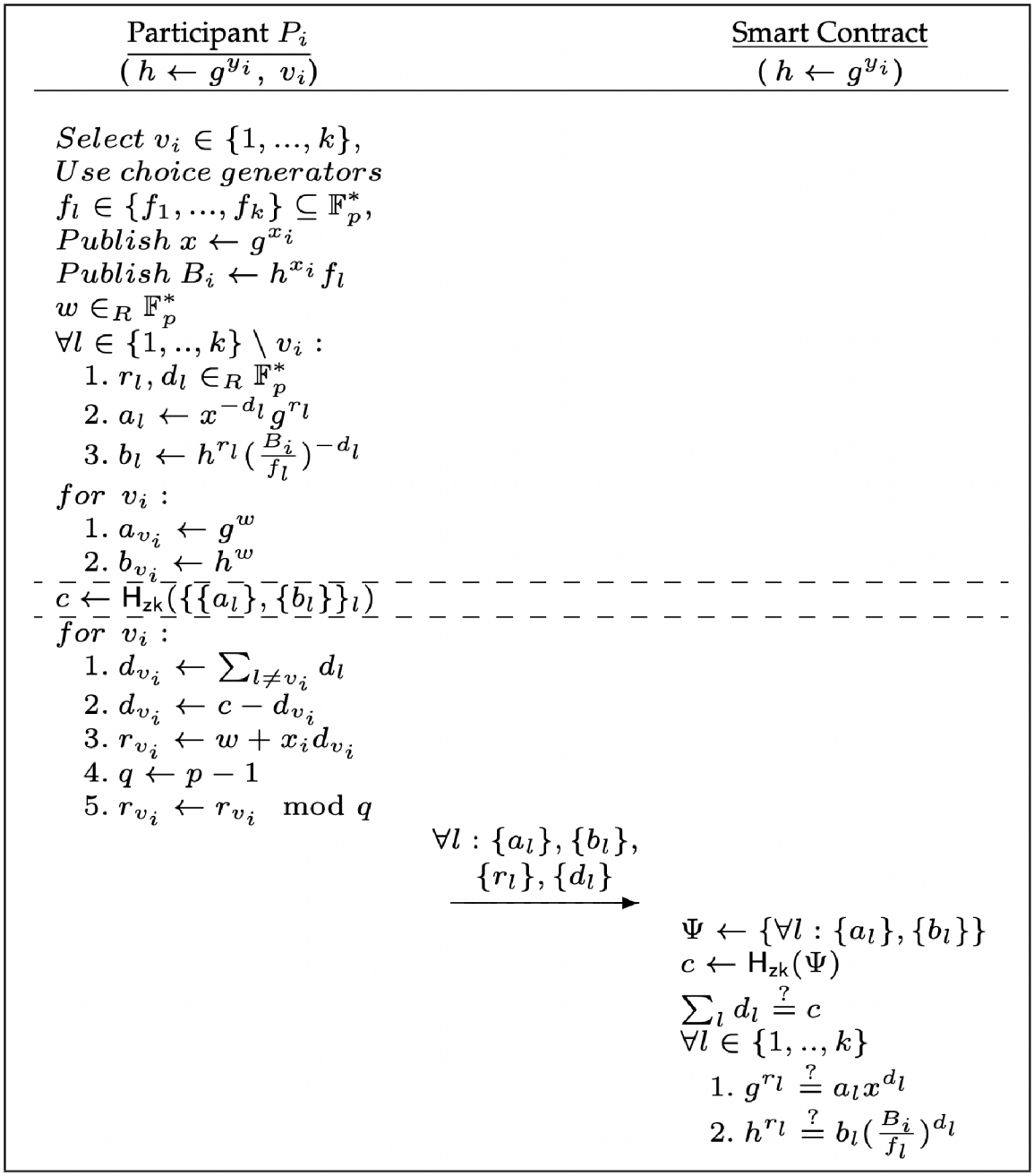


Figure 4.3: Zero knowledge proof of set membership for 1-out-of-$k$ choices.

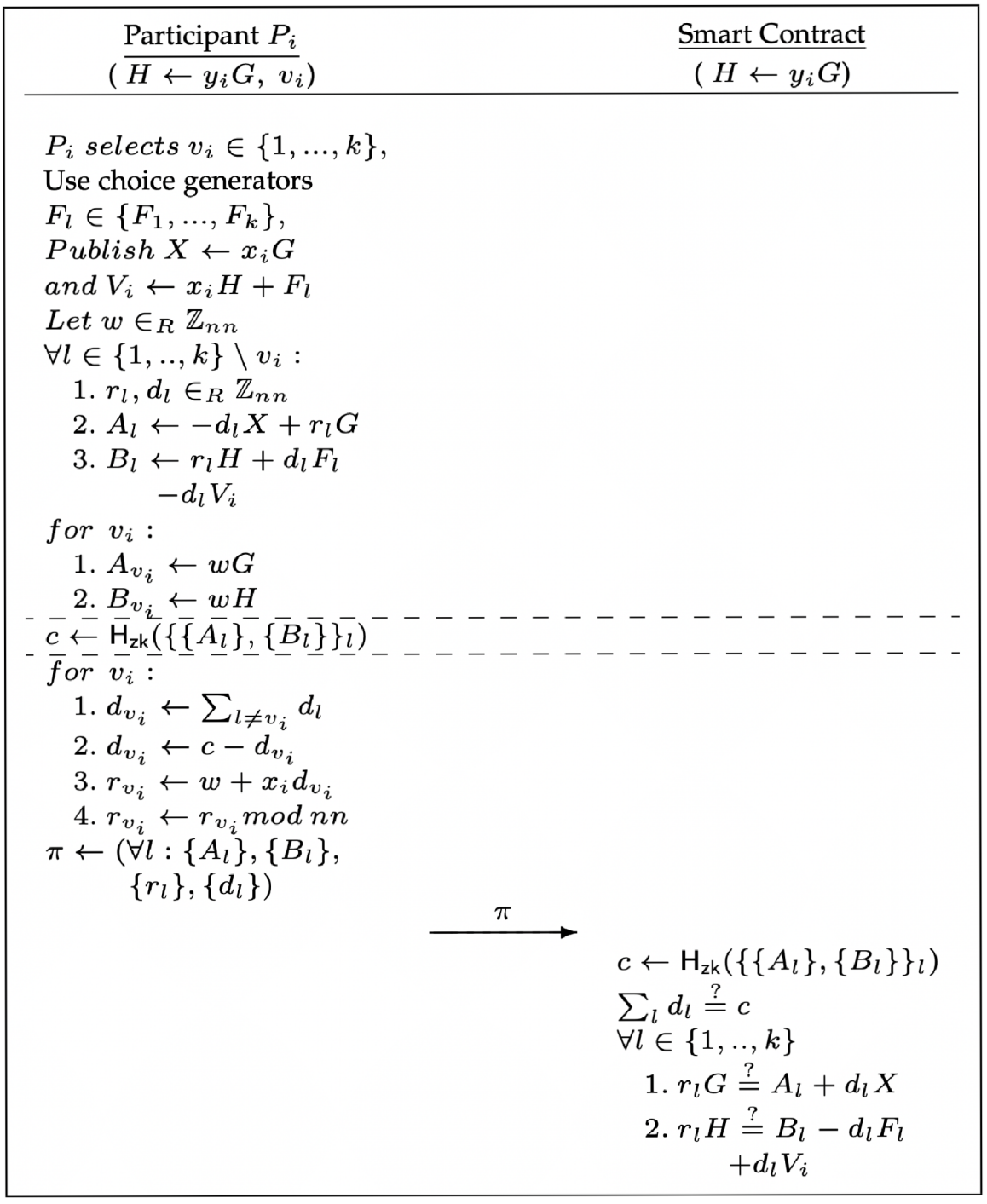


Figure 4.4: ZKP of membership for 1-out-of-$k$ choices on the elliptic curve.

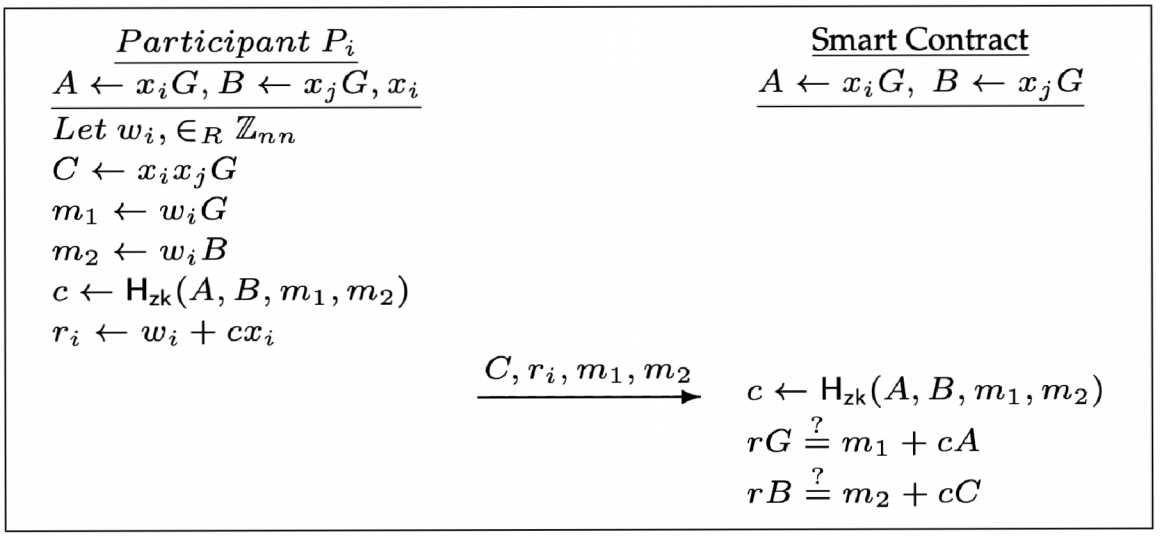


Figure 4.5: ZKP verifying correspondence of $x_ix_jG$ to PKs $A = x_iG, B = x_jG$.

# Chapter 5

# BBB-Voting on Solana design

The goal of this work was to create a working prototype of a voting system based on the BBB-Voting algorithm that will run on the Solana blockchain. Based on this prototype, it should be possible to determine whether further development makes sense for Solana, based on three criteria. The work started with the assumption that BBB-Voting can be ported to the Solana blockchain, that implementing voting on Solana will be cheaper and, most importantly, that it will be theoretically possible to perform large votes due to Solana's high throughput. This is because Ethereum can only process around 15 transactions per second and so voting millions of people would take on the order of weeks of Ethereum's computing time.

A variant using integer arithmetic was chosen for development because it is more straightforward and thus more easily transferable from Ethereum to another technology. The provided BBB-Voting implementations of Ethereum smart contracts included extended capabilities. One of these is the ability to restart an erroneous vote and punish the participant who caused the error. The second is the timing of the different phases with the possibility to punish participants or the authority who do not perform the corresponding action within the specified time. These functions have been omitted from the BBB-Voting proposal for Solana because they are irrelevant to the achievement of the project's goal.

Thus, the functions of the proposed system correspond to the basic voting use case, which includes the functionality of the methods `constructor`, `enrollVoters`, `submitEphemeralPK`, `computeMPCKeys`, `submitVote` and `computeTally` described in the previous section.

## 5.1 Smart contract design

Smart contracts on Ethereum and Solana are fundamentally different. On Ethereum, both the data and the functional code are stored together, whereas on Solana, the program – the executable code – is a special type of account that cannot contain any other data. Data on the Solana blockchain is kept in separate data accounts distinct from the accounts containing the executable program. This approach allows having the executable deployed only once and using it equivalently to multiple deployed contracts on Ethereum. Thus, the provided implementation for Ethereum had to be split into a data part and a functional part. One voting will be represented by corresponding data accounts, and the deployed executable part of SC will be common to all votings.

As far as data is concerned, in addition to candidates and voters' addresses, the smart contract must also store the basic voting parameters, the data received from voters, the calculated MCP keys, and the results. The basic voting parameters are the appropriate modulo used in computations, the generators for each candidate, the generator for the voters. The data received from voters are ephemeral PK and blinded vote.

There are several ways to store the data of a specific voting within the data accounts. All data can be stored within a single account, or the data can be split across multiple accounts, either by type, e.g. one account for ephemeral PKs, one for MPC keys, etc., or by affiliation to a given voter. The second approach, splitting the data into multiple accounts, is advantageous for increasing the number of possible voters, as the data account on Solana has a limited size (10 MB at the time of writing). However, for the purposes of this project, the simpler approach of storing all data in one account is sufficient. So each vote will be represented by a single data account that will contain all the voting data.

For the executable part of the voting smart contract, it is similarly possible to store the individual methods in a single program account, or to split them into several. Nevertheless, for the purposes of this project, it will likewise be sufficient to store all the necessary functions in a single program account.

The final prototype will have the following design: A single program account containing all the methods necessary for the BBB-Voting algorithm will be deployed on the Solana blockchain. The creation of a new voting will consist of initializing a new data account. The initialization will be performed using a method found in the program account. Thus, the data account will be owned by the voting program and only that program will be able to modify its data. The design of the voting system is shown in Figure 5.1.

Voting will have 5 phases, where for each phase it will be possible to invoke the desired functionality. The first of these will be `CONFIRMING`, where the vote will be immediately after it is initialized. In this phase, all participants will need to confirm their participation in the vote by sending their ephemeral public key to the smart contract. Once all have done so, the voting will enter the `CONFIRMED` phase. In this phase, the authority must initiate the vote by invoking the MPC key computation. Then the voting will enter the `VOTING` phase in which the voters will send their blinded votes to the contract. Once all voters have voted, the voting will enter the `VOTED` state. It will be in this state until the authority sends the correct tally to the VSC. Then the vote will enter the `DONE` phase, which will be its final phase in which no further interaction will be possible.

The program will work with 64-bit keys. For this project, a variant of the voting algorithm working with integer arithmetic was chosen. The corresponding provided implementation worked with 128-bit keys, since the largest natively supported numbers in Solidity are 256-bit. In Rust, the largest natively supported numbers are 128-bit numbers and thus working with larger than 64-bit keys could not be implemented natively, which would make the solution unnecessarily complicated.

### 5.1.1 Program account

The program account of the smart contract will include the subsequent methods:

- `init_voting`: This method will provide functionality for initializing the data account. It will be equivalent to the `constructor` and `enrollVoters` method of the sample implementation. Thus, its parameters will be: a vector of names of all candidates, a vector of wallet addresses of all voters, a modulo, a generator for voters, and generators for each candidate. The result will be an initialized new data account representing

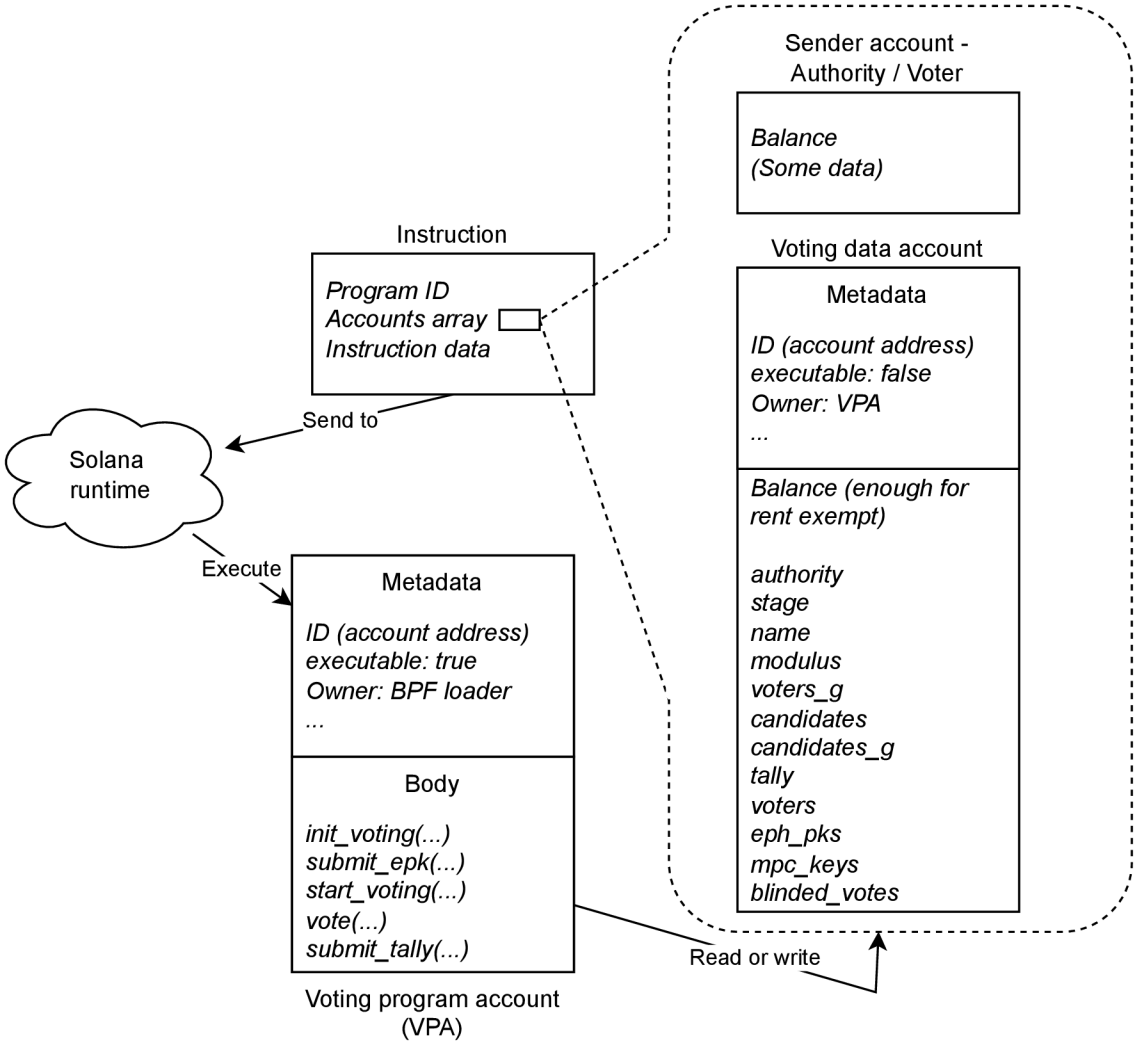


Figure 5.1: Phases of the BBB-Voting protocol [29].

the given vote containing the data passed to this method with the stage attribute set to `CONFIRMING`.

- `submit_epk`: This method will be used for voters to send their public ephemeral key to the VSC. It will therefore have a single parameter, namely the ephemeral key. When all voters have sent their key, this method will set the voting phase to `CONFIRMED`.

  The method `submit_epk` will only be available in the `CONFIRMING` voting phase.

- `start_voting`: This method will be used to calculate the MPC keys. It will not have any parameter because the contract will already contain all the data needed to calculate the MPC. This method will be called by the authority and after it is completed, the voting phase will be `VOTING`.

  The method `start_voting` will only be available in the `CONFIRMED` voting phase.

- `vote`: This method will allow voters to send in their blanked votes to the VSC. It will have five parameters – blinded vote and four NIZK proof components. The method will verify the validity of the NIZK proof and, if successful, will store the submitted blinded vote in the VSC. When the last voter's vote is processed correctly, the method will change the VSC phase to `VOTED`.

  The method `vote` will only be available in the `VOTING` voting phase.

- `submit_tally`: This method will be used to write the results into the VSC. The method will be called by the authority and will have one paremeter, which will be a vector of the number of votes received for each candidate. The method will verify the correctness of the results received and, if correct, will store the results in the VSC. It will then move the voting to its final `DONE` phase.

  The method `submit_tally` will only be available in the `VOTING` voting phase.

### 5.1.2 Data account

The data account of the smart contract will include the following data:

- `authority`: The address of the authority that initiated the vote. This is necessary to prevent others from calling methods intended for the authority.
- `stage`: The current voting phase to simplify a smooth step-by-step transition through the voting algorithm.
- `name`: A name of the voting that will allow it to be more conveniently identified.
- `modulus`: The modulus that will be used in the calculations in the algorithm.
- `voters_g`: A common generator for all voters.
- `candidates`: A vector of candidate names.
- `candidates_g`: Generators of individual candidates.
- `tally`: Vector of voting results.

- `voters`: A vector of voters' wallet addresses. It will allow voters to be identified and thus prevent others from getting involved. In addition, it will also determine the index of each voter, which is needed in the calculations of the algorithm.
- `eph_pks`: Vector of ephemeral keys received from voters. A key at a particular index in this vector belongs to a voter whose wallet address is at the same index in the `voters` vector.
- `mpc_keys`: Vector of computed MPC keys. A key at a particular index in this vector belongs to a voter whose wallet address is at the same index in the `voters` vector.
- `blinded_votes`: Vector of blinded votes received from voters. A key at a particular index in this vector belongs to a voter whose wallet address is at the same index in the `voters` vector.

# Chapter 6

# Implementation

The Anchor framework was used to develop the final prototype. Anchor is a Rust framework that makes it faster and safer to create Solana programs. Anchor increases the speed of development by including boilerplate features such as (de)serialization of accounts and instruction data. It increases security by automatically doing certain security checks instead of the developer. It also makes it easy and concise to define additional security controls separately from the actual business logic of the program. The final prototype is thus programmed in the Rust language – Rust version 1.59.0 and Anchor framework version 0.24.0 were used.

## 6.1 Data account

The data account described in detail in the previous chapter in Section 5.1.2 is defined using Anchor as shown in Listing 6.1.

As for the account size, it is calculated as follows: each account created with Anchor needs 8 bytes of Anchor's internal `discriminator`, `i128` needs 16 bytes, `Pubkey` needs 32 bytes, the `Vec` structure needs 4 bytes, `String` needs 4 plus the number of characters times 2. This means that the voting data account needs: for the basic data 8 (`discriminator`) plus 188 (`modulus`, `voters_g`, `stage` – max 10 characters, `name` – max 20 characters, 7 `Vec` structures), for each candidate 76 bytes (`generator`, `tally`, `name` – max 20 characters), and for each voter 84 bytes (`address`, `eph_pk`, `mpc_key`, `blinded_vote`). Thus, for example, for a voting with three candidates and five voters, it is 844 bytes in total.

## 6.2 Anchor Contexts

On Solana it is necessary that a transaction that invokes a program account method includes all data accounts that will be used by that method. n Anchor this is done by the first parameter of each method, which is of type Context. Context is a generic type over the Accounts structure. The Accounts structure is in turn a way to define the group of accounts that will be used by a particular method. Various constraints and safety checks can also be defined within this structure.

For the final system it was necessary to define 3 structures of Accounts. One for the methods that will be called by the authority, one for the methods that will be called by the voters and the third for the `init_voting` method. The last mentioned is shown in Listing 6.2.

```rust
#[account]
pub struct VotingState {
    pub authority: Pubkey,
    stage: String,
    name: String,
    modulus: i128,
    voters_g: i128,

    candidates: Vec<String>,
    candidates_g: Vec<i128>,
    tally: Vec<i128>,

    voters: Vec<Pubkey>,
    eph_pks: Vec<i128>,
    mpc_keys: Vec<i128>,
    blinded_votes: Vec<i128>,
}
```

Listing 6.1: Code defining the voting data account structure.

```rust
#[derive(Accounts)]
pub struct InitVoting<'info> {
    #[account(init, payer = authority, space = 152 + 3 * 72+ 9 * 52 )]
    pub voting_account: Account<'info, VotingState>,
    #[account(mut)]
    pub authority: Signer<'info>,
    pub system_program: Program<'info, System>,
}
```

Listing 6.2: Accounts structure that is used when creating a new voting.

```rust
#[derive(Accounts)]
pub struct ByVoter<'info> {
    #[account(mut)]
    pub voting_account: Account<'info, VotingState>,
    pub authority: Signer<'info>,
}

#[derive(Accounts)]
pub struct ByAuthority<'info> {
    #[account(mut, has_one = authority)]
    pub voting_account: Account<'info, VotingState>,
    pub authority: Signer<'info>,
}
```

Listing 6.3: Accounts structures for voting program methods.

The name of this structure is InitVoting. The first parameter of the `init_voting` method is therefore of type Context⟨InitVoting⟩.

There are three accounts in the structure that the `init_voting` method will work with. The first is `voting_account`. This account is in a structure with three constraints. Constraint init means that this account will be created when calling a method that has the initVoting structure as a parameter. The created account will be rent-extempt and will be paid for by the account defined by the constraint payer. This account must be a signer, that is, one of the senders of the transaction. The constraint space specifies how many bytes will be allocated for the created account.

The second is authority. This one has a constraint `mut`, which means that its content will be changed. In this case, this will be the number of lamports that the account owns – it will pay for the initialization of the voting data account.

The third is system_program, which is required in the account validation struct whenever an init constraint is present. This is because SystemProgram is the program used to create accounts. And since a new account will be created in the transaction in which the Account validation structure is used, SystemProgram, which is also a program account, will need to be used, and any account that will be used must be listed in the transaction.

The other two Accounts structures used within the voting system are very similar and are shown in Listing 6.3.

The two structures above contain identically two accounts each. The first one is the account of the given voting – `voting_account`. It is always present with constraint `mut`, since all methods of the voting program will modify its contents. The second account is named authority and it is the account of the signer of the transaction within which some method using the given contexts is called. The only difference is that the `ByAuthority` structure, which will be a parameter to the methods intended for the authority, contains the `has_one` = authority constraint above the `voting_account` constraint. This constraint verifies a match between the account named authority within the `ByAuthority` structure and the authority attribute within the data account it applies to – in this case, the `voting_account` account. If they do not match, an error will occur and the transaction will not take place. This ensures that no other party besides the authority can use the methods intended for the authority.

## 6.3 Program account

The final prototype contains 5 methods whose purpose and desired behavior are described in Section 5.1.1. The first is the `init_voting` method, which has 7 parameters. The first one is the `InitVoting` Accounts structure wrapped in a generic Context class. This will ensure the creation of a new data account for voting. The other parameters are name, module, voters_g, candidates_g, candidates, voters, which is a subset of the data contained in the data account and a description of each item is given in Section 5.1.2. The body of the method consists of verifying that no duplicate candidates or voters have been provided and assigning each data account attribute its initial value.

The second method is submit_epk. This method has two parameters. The first one is the `ByVoter` Accounts structure wrapped in a generic Context class, as this method serves for all voters. The second parameter is the ephemeral key of the subscriber. The method writes this key to the `eph_pks` vector in the vote data account at the appropriate index – the sender index in the voters vector.

The third method in the final prototype is the `start_voting` method. It has a single parameter and that is `ByAuthority` Accounts structure wrapped in a generic Context class. The purpose of this method is to compute MPC keys after all voters have sent their ephemeral public key to the smart contract. It does this according to Equation 4.2 and the main calculation code looks is shown in Listing 6.4;

In the calculation, the auxiliary functions `mul_mod`, which has three parameters - x, y and k - and results in $(x * y)\%k$, and `inv_mod`, which is a function for calculating the modular multiplicative inverse, are used.

The fourth method is the vote method, which takes 6 parameters. The first is `ByVoter` Accounts structure wrapped in a generic Context class, as this method serves for all voters. The fourth method is the vote method, which takes 6 parameters. The second parameter is blinded vote and the other 4 are the 4 parts of the NIZK proof. Specifically, these are `proof_a`, `proof_b`, `proof_r`, `proof_d`, and the method verifies their correctness according to the algorithm shown in 4.3. The main verification code looks is shown at Listing 6.5.

```
for i in 0..voters_cnt {
    let mut numerator: i128 = 1;
    for j in 0..i {
        numerator = mul_mod(
            numerator,
            voting_account.eph_pks[j],
            voting_account.modulus
        );
    }

    let mut denominator: i128 = 1;
    for j in i + 1..voters_cnt {
        denominator = mul_mod(
            denominator,
            voting_account.eph_pks[j],
            voting_account.modulus,
        );
    }
    let inv_mod_denominator = inv_mod(
        denominator,
        voting_account.modulus
    );
    let mpc_key = mul_mod(
        numerator,
        inv_mod_denominator,
        voting_account.modulus
    );
    voting_account.mpc_keys.push(mpc_key);
}
```

Listing 6.4: Computation of MPC keys according to Equation 4.2.

```
for l in 0..voting_account.candidates.len() {
    let mut left = mod_exp(
        voting_account.voters_g,
        proof_r[l],
        voting_account.modulus
    );
    let mut right = mod_exp(
        voting_account.eph_pks[index],
        proof_d[l],
        voting_account.modulus,
    );
    right = mul_mod(right, proof_a[l], voting_account.modulus);

    if left != right {
        return Err(error!(ErrorCode::InvalidBlindedvoteProofs));
    }
    left = mod_exp(
        voting_account.mpc_keys[index],
        proof_r[l],
        voting_account.modulus,
    );
    right = mul_mod(
        blinded_vote,
        inv_mod(voting_account.candidates_g[l], voting_account.modulus),
        voting_account.modulus,
    );
    right = mod_exp(right, proof_d[l], voting_account.modulus);
    right = mul_mod(right, proof_b[l], voting_account.modulus);
    if left != right {
        return Err(error!(ErrorCode::InvalidBlindedvoteProofs));
    }
}
```

Listing 6.5: Verification of submitted NIZK proof according to Equation 4.3.

In addition to the functions `mul_mod` and `inv_mod` from the previous algorithm, this algorithm uses the function `mod_exp`, which serves to calculate the modular exponentiation.

The last method of the final prototype is the `submit_tally` method for sending the results to smart contract. The method has 2 parameters, the first is first one is the `ByVoter` Accounts structure wrapped in a generic `Context` class, as it has no benefit to limit who sends the correct result to the smart contact. That is because the method itself verifies whether the result sent is correct, according to formula 4.6. Code for this operation is shown in Listing 6.6.

After the correct result is sent to the smart contract using the `submit_tally` method, the voting phase in the voting data account is set to `DONE` and no more methods can be invoked over this account.

```
let left = prod_mod(blinded_votes, voting_account.modulus);
let mut right = 1;
for l in 0..voting_account.candidates.len() {
    right = (right
        * mod_exp(
            voting_account.candidates_g[l],
            tally[l],
            voting_account.modulus,
        ))
        % voting_account.modulus;
}

if left != right {
    return Err(error!(ErrorCode::InvalidTally));
}

voting_account.tally = tally;
```

Listing 6.6: Verification of submitted tally according to Equation 4.6.

# Chapter 7

# Front-end for voting

As part of this project, a working prototype of a web application was also created, allowing the use of the developed voting system in a web browser.

## 7.1 Design

A basic use-case has been specified for the prototype web application and that is a simple vote. Voting consists of creating a poll, running the poll and displaying the results. The design of the application thus includes everything necessary to perform these phases.

Two options were considered, either to split the application and make two applications, one for creating and managing the votes and the other for voting in them, or to create one application that provides all the options. Since it was assumed that it would be possible to detect both the address of the authority and all legitimate participants, and thus it would be relatively easy to detect which controls should be available to the user, the option of implementing a single application won out. The use case diagram of the web application is shown in Figure 7.1.

### 7.1.1 Creation of voting

The app is designed in such a way that it offers the possibility to create a new poll or return to previous polls from the home screen. In order to create a vote, as well as to otherwise work with the app, you need to be connected to the app using the Solana wallet Phantom wallet[1].

To create a vote, the user has to enter its name and confirm it. This will create a new account on the blockchain representing that vote. The addresses of all created votes are stored in persistent local storage in the browser – the implementation uses React-Redux and Rexus-Persist for this.

### 7.1.2 Creation of voting

The application has been designed so that from the creation of a vote to its completion, its management by the authority takes place within a single screen. This is where the status messages of the vote change dynamically, as well as the options that the authority has at that moment. Similarly for the voting participant.

[1] https://phantom.app/

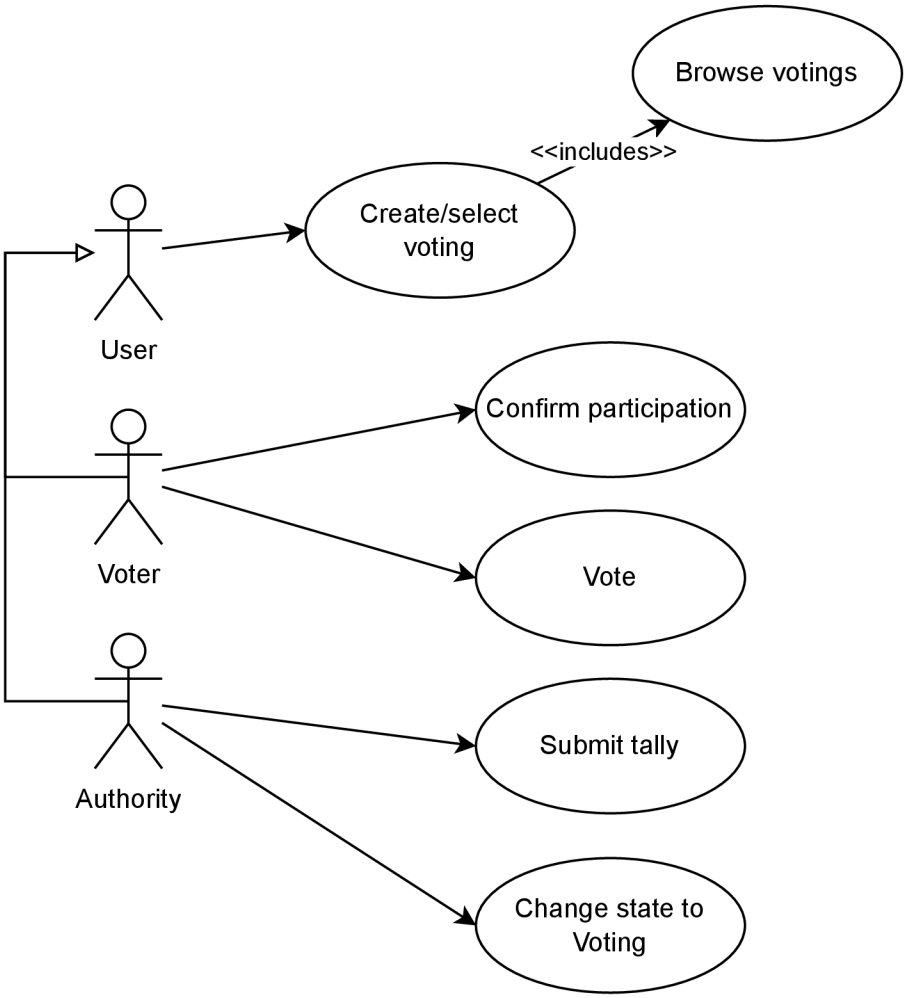


Figure 7.1: Use case diagram of the voting web application.

### 7.1.3 Voting process and results

The proposed application sequentially takes the authority or voting participant through states reflecting the BBB-Voting protocol states described in section 4.2.1:

1. **Registration:** The authority has the possibility to enter all verified wallet addresses of the participants and also of all voting candidates into the created voting account. The participant has no options at this stage.

2. **Setup:** The authority has no options at this stage, it just waits for all participants to take the necessary actions. The participant has the option to confirm his participation in the election (in the BBB-Voting protocol, the participant creates and sends his ephemeral key in this step).

3. **Pre-Voting:** At this stage, the authority has the opportunity to start the voting (in the BBB-Voting protocol, in this step, the authority computes and sends the key to the MPC voting account). The participant has to wait for this step.

4. **Voting:** The authority has no options at this stage, it just waits for all participants to take the necessary steps. Participants have the option to vote for their candidate.

5. **Tally:** In this last phase, all participants have the opportunity to see the results of the vote.

If the authority is also a voting participant, he does not need to switch screens. He can see everything he needs within a single screen and therefore also has the participant user elements at his disposal.

### 7.1.4 Implementation

The JavaScript framework React[2] was used for the implementation. So the whole application is a single-page of sorts, and since it's decentralized – working with the blockchain and not a server or database – all it takes is a single query to the server to get the code for the page. Otherwise, everything else runs on the client device.

To work with the blockchain, the app uses the JS libraries '@solana/web3.js' and '@project-serum/anchor' with which it can connect to the user's wallet and also send transactions to the blockchain. A typical use case assumption is that the user has a software wallet, the Phantom wallet, installed in the browser.

The entire application code is written in JavaScript in the React framework and currently consists of about 1800 lines of authoring code. In addition to the React framework, the JavaScript library for React teeming Material UI[3] has been used extensively during development. In addition, React-Redux is used in the application for state management and persistence of some data.

### 7.1.5 Final web app

The final prototype offers complete options for creating a poll, voting and displaying the results. It is also responsive (see image 7.5) to make the app easy to work with on both desktop and mobile.

[2] https://reactjs.org/
[3] https://material-ui.com/

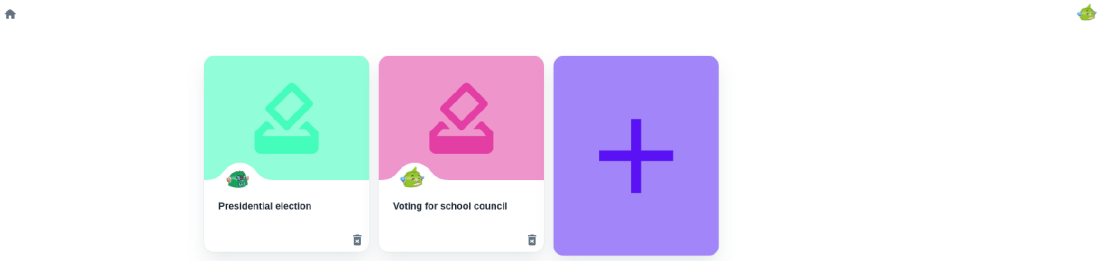


Figure 7.2: A page for selecting or creating a new voting.

As can be seen in the image 7.2, the application provides the possibility to create a new poll or to use a previously created poll whose address is located in the Local Storage. Clicking on the created poll takes the user to the poll management page, a sample of which is shown in the 7.3 image. The image shows a page displaying a specific vote in the second phase from the authority's perspective. If the user wants to participate in the voting he needs to find out the address of his contract. The image 7.4 shows the voting from the participant's point of view.

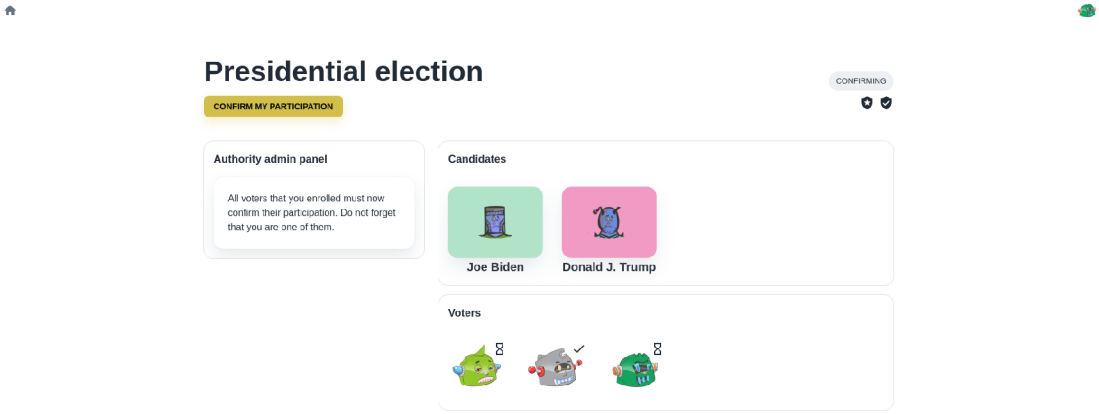


Figure 7.3: Voting management page, captured in the second phase of the protocol.

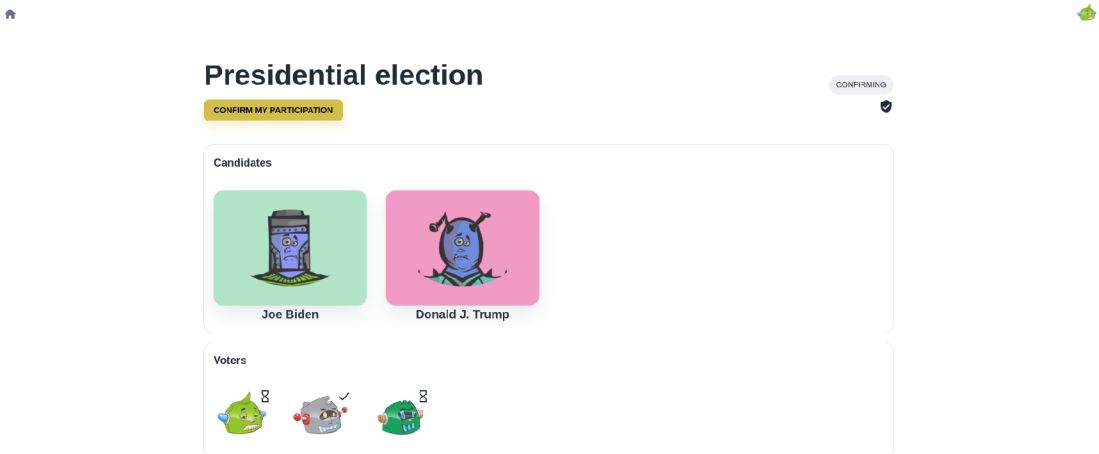


Figure 7.4: Voting participation page. Captured in the second phase of the protocol.

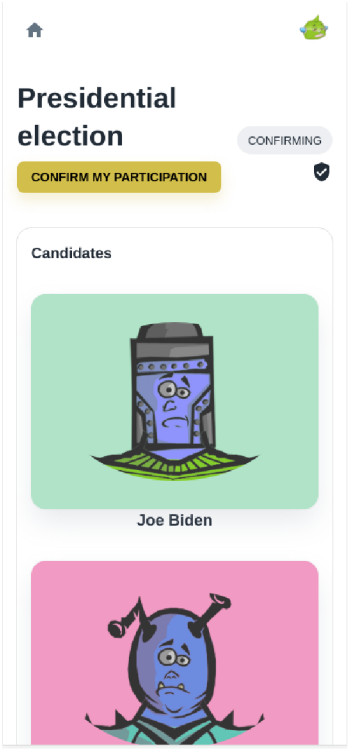


Figure 7.5: View the voting participation page on a mobile device.

# Chapter 8

# Results

The final prototype of the BBB-Voting system for Solana was created by programming the Rust module as described in the previous two chapters. The module was named `so_vote` (which is an ambiguous name – it means „so vote“ as in „it's easy cheap and secure, so vote“ and it also is an abbreviation for Solana vote).

## 8.1 Tests

To test the final system, 5 tests were written using Mocha – an open-source JavaScript test framework that runs on Node.js. These tests comprehensively test the developed system by simulating the voting process from start to finish. For each method of the final prototype, there is one test that tests whether the desired use-case works. Thus, the tests involve authority and a configurable number of participants and candidates.

The first test consists of the creation of a poll by the authority and then verifying that the poll exists and has the desired values set. The second test is to confirm the participation of each voter and verify, based on the voter index, that the appropriate ephemeral public keys are present in the SC at the appropriate indices in the `eph_pks` vector in the voting data account. The test also involves verifying that the VSC has automatically passed to the next phase, in this case „CONFIRMED “. Each subsequent test also contains such a verification of the transition to the next phase appropriate for the test.

The third test verifies the ability of the authority to initiate a vote and also the correctness of the calculation of MPC keys within the smart contract. Verification of the correctness of the MPC keys is made possible by comparing their value and the value calculated by the off-chain implementation of this calculation, which was provided in JavaScript for the development of this project.

The fourth test is for all voters to vote. Using the JS object provided for the development of this project, the blinded votes and the corresponding NIZK proofs of each participant are obtained and sent to the smart contract. The expected behavior is that it accepts them – this verifies the correctness of its implementation regarding the verification of NIZK proofs. This is because their computation is assumed to be correct – since it is done using a well-tested implementation of NIZK proof generation.

The fifth test tests whether the VSC accepts the correct voting outcome and also that it rejects all other possible outcomes. To generate all possible results, a recursive function `allPossibleTallyCombinations` has been implemented which takes two parameters namely the number of voters and the number of candidates.

## 8.2 Security

The final prototype cannot be considered secure in terms of voting privacy, as because it uses cryptography for breaking which it is necessary to solve the discrete logarithm problem. This is considered secure when using at least 1024-bit keys. The final prototype works with 64-bit keys and is thus vulnerable to vote compromise.

However, this is not a big problem because this work is concerned with creating a proof of concept BBB-Voting system for the Solana blockchain. The implementation could be modified for larger and thus more secure keys, but it would have to be done in a different way than it would be done natively in Rust. Due to the fact that there are bigger obstacles for the final prototype to be realistically deployable, this security issue was not addressed within the project.

Another threat to the smooth running of potential elections, however, appears to be the Solana blockchain itself. There have been several occasions when the network has experienced problems and remained offline for several hours. An example is the DoS attack that took place on 14 September 2021 [10], which also caused the network to fork. Another similar outage occurred on 4 January 2022 [9]. Grayscale Investments wrote „There may be flaws in the cryptography underlying the network, including flaws that affect the functionality of the Solana Network or make the network vulnerable to attack. Solana's economic incentives may not function as intended, which may cause the network to be insecure or perform sub-optimally“ [11]. These suspicions about Solana's vulnerability are partly caused by to the fact that the Solana Network selects one exclusive propagator of transactions for a certain period of time, and this information is known in advance – the promoter may become the target of an attack, and this attack will affect the entire network. This is similar to one of the risks that blockchain technology was supposed to eliminate compared to centralized systems – single point of failure.

In addition, some people are also wary of trusting Solana because it has a relatively low degree of decentralization – it recently reached 1,000 validators for the first time [26]. That's significantly less than Ethereum, which has hundreds of thousands of miners [16]. So in theory, it wouldn't be that hard for a group of Solana validators to collude and commit malicious activity on the network. At least it would be much easier than on Ethereum.

On the other hand, Solana's CEO claims that the outages were not caused by attacks, but by other things, such as trading bot activity or problems in processing certain transactions. These are bugs that can be fixed and it's better that they have shown themselves now than later. [9] Also, the number of Solana validators is still growing. So to say that Solana is an unsuitable blockchain in terms of security for elections would be premature.

## 8.3 Computational demand

During development and testing, it was found that the implementation of the vote method, where the validity of the NIZK proof must be verified, is very computationally intensive. Solana is limited in how many so-called compute units a program call can consume. At the time of writing this thesis this limit is 200,000. This limit was not sufficient for the vote method and its call always ended with error.

For the development and testing of the program it was therefore necessary to get rid of this limit. This was achieved by downloading and modifying the Solana source code and its subsequent custom installation. In this way, the limit was raised to 5,000,000 compute units. The compiled Solana test validator could not be run when trying to raise the limit.

Since an in-depth examination of the Solana blockchain source code is beyond the scope of this project, no other attempts were made and further work was done with a local validator with a limit of 5,000,000 compute units.

It was found that single call to the vote method consumes approximately 2,500,000[1] compute units for a voting that has two candidates and approximately 3,800,000[2] compute units for a voting that has three candidates. With four candidates, even the 5,000,000 limit was no longer enough.

For testing and prediction purposes, an implementation of the algorithm using 32-bit keys was also created. Here the computational power requirement of the vote method was much lower - for a two-candidate vote it was approximately 284,000[3] compute units and for a three-candidate vote it was 412,000[4] compute units. This is approximately 9 times less than using 64-bit keys. Such a big difference can be explained by the fact that today's processors are 64-bit, and with 32-bit keys the results of multiplication (64-bit numbers) can be processed more optimally by the processor. Thus, it can be assumed that just between using 32-bit and 64-bit keys there is a step difference in the computational complexity of the algorithm, and with further increases the complexity would not grow so sharply. However, since no implementation has been created for either smaller than 32-bit or larger than 64-bit keys, it is not possible to state this with certainty.

Anyway, the difficulty seems to increase linearly with the number of candidates, which is as expected, since for each additional candidate another iteration of the loop in the vote method is needed. This trend is illustrated in Figure 8.1, which shows the dependence of computational complexity on the number of candidates when using 32-bit keys. Figure 8.2 shows the dependence of computational intensity on the number of candidates when using 64-bit keys. Graph 8.3, in turn, shows the prediction of the further evolution of the dependence of computational complexity on the number of candidates when using 64-bit keys. To create this graph, new points were created by projecting the growth trend from Graph 8.1, as further measurements were not possible.

## 8.4 Discussion on the suitability of the final prototype

The aim of this work was to analyze the suitability of the Solana blockchain for BBB-Voting and then to implement this voting system. This is because the BBB-Voting protocol has so far only been implemented for Ethereum, which is insufficient in speed for large-scale voting. Solana is significantly faster in terms of throughput compared to Ethereum. However, in developing the system for Solana, a fundamental problem emerged. Solana is limited by the number of computing units that a single program invocation can consume, and the prototype created does not meet this limit.

However, the prototype can be used locally using a specially adapted solana-test-validator that was created for this work. In addition, it can be assumed that Solana's computing budget will increase over time. At the time of writing, the Solana documentation states „The compute budget currently applies per-instruction but is moving towards a per-transaction model.“. This means that there will be changes to Solana's compute budget in the future, as Solana is a relatively new and evolving project. Just moving to a per-transaction model would be getting closer to the real deployability of this project, be-

[1] In 30 experiments, the mean was measured to be 2,543,705 and the standard deviation was 52,073.
[2] In 30 experiments, the mean was measured to be 3,816,487 and the standard deviation was 65,041.
[3] In 30 experiments, the mean was measured to be 283,947 and the standard deviation was 12043.
[4] In 30 experiments, the mean was measured to be 412,492 and the standard deviation was 16,693.

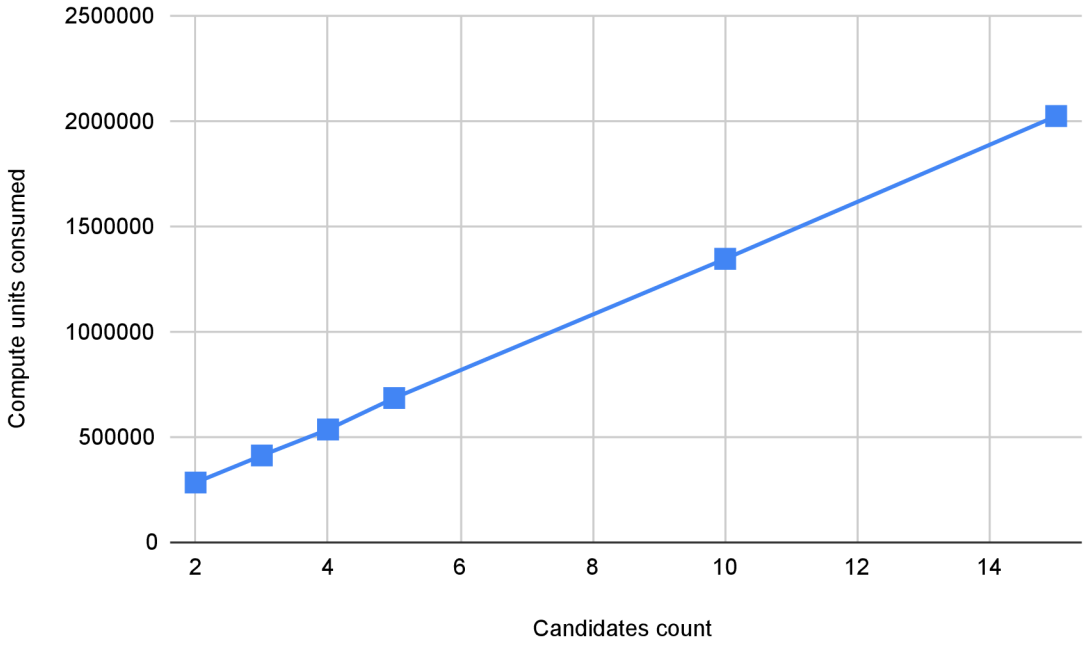


Figure 8.1: Dependence of computational complexity of vote method on the number of candidates when using 32-bit keys.

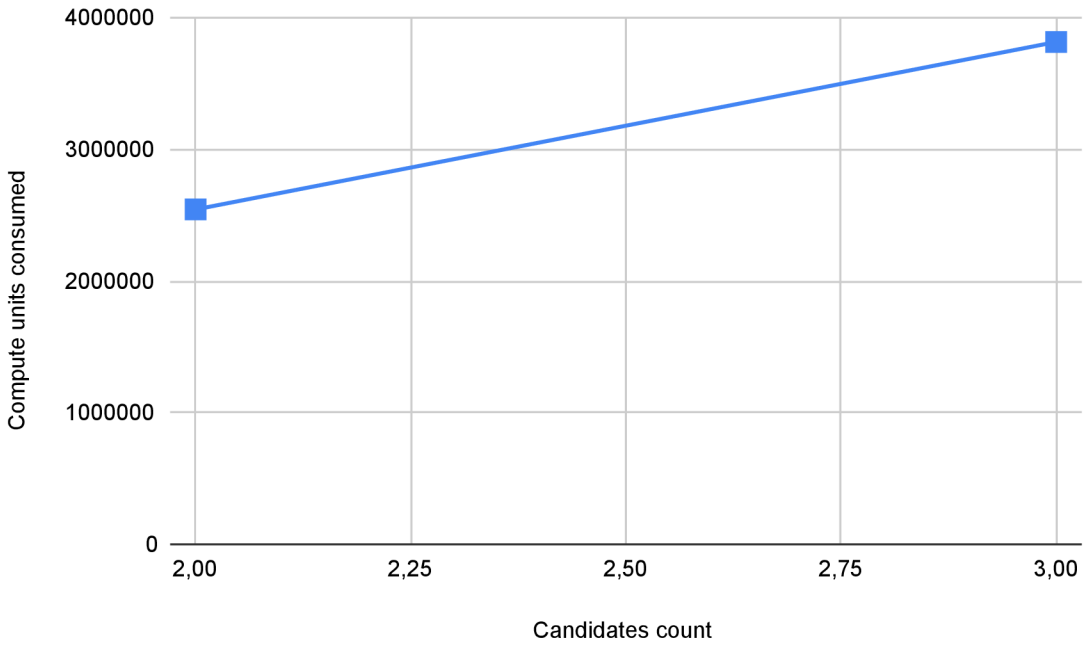


Figure 8.2: Dependence of computational complexity of vote method on the number of candidates when using 64-bit keys.

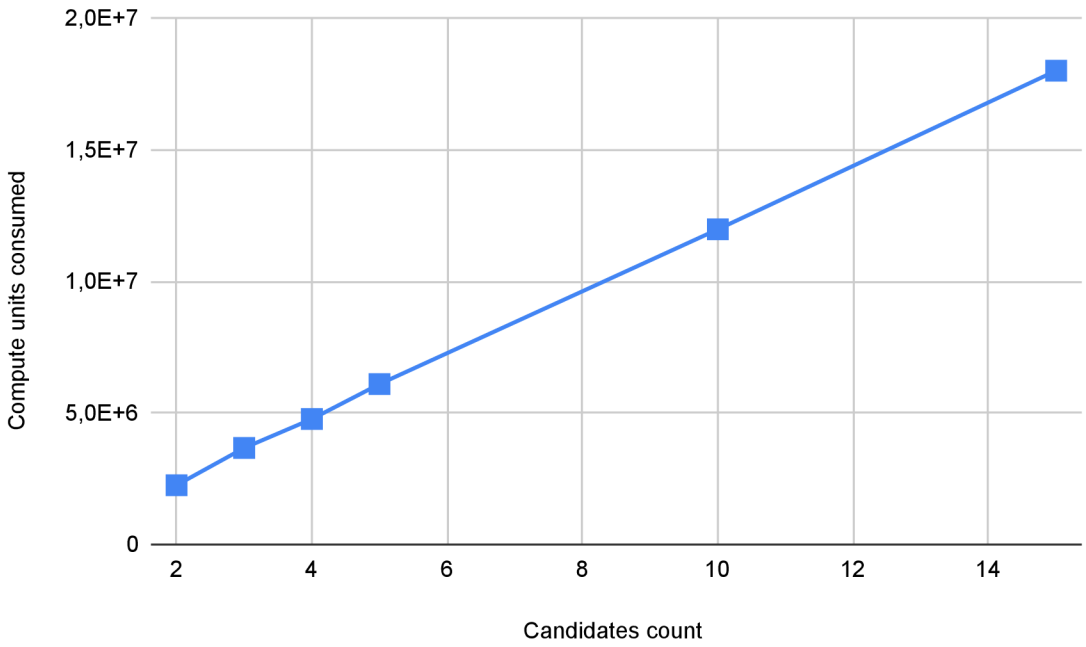


Figure 8.3: Dependence of computational complexity of vote method on the number of candidates when using 64-bit keys.

cause in the case of transactions it is possible to request a special instruction to increase the budget for the whole transaction. Nevertheless, this is not possible for a single instruction, so in the current state the prototype cannot be used on the Solana mainnet.

Therefore, it is not possible to test the developed voting system on the Solana mainnet. Also, performance testing on a Solana local validator would have no meaningful value. However, one can make a rough estimate of how usable the final prototype would be if Solana's computational budget were increased. Solana handled performance peaks of 65,000 transactions per second. The theoretical limit in the future is estimated to be 710,000 transactions per second. [19] Currently, the compute budget for a transaction is 200,000 compute units. If Solana can handle 65,000 transactions per second and each transaction can consume 200,000 compute units, it appears that Solana can handle 13,000,000,000 compute units per second. This would mean that Solana could handle 3421 invocations of the critical vote method, which needs about 3,800,000 compute units to verify the NIZK proof when voting from 3 candidates. Thus 295,574,400 participants would be able to vote in a voting during one day. If the theoretical limit of 710,000 is taken into account 3,228,581,907 voters could vote per day.

Of course these are rough estimates, but they look promising. The BBB-Voting implementation for Ethereum does fit into the compute budget (on Ethereum it's called the gas limit), but Ethereum's throughput is 16 transactions per second and the estimated 3421 votes per second is a 2138.125 % improvement. Nevertheless, it can be assumed that granting voting privacy by using at least 1024-bit keys would increase the computational requirements of the program. To deploy it, Solana's compute budget would have to be increased even more and the number of voters that could vote in one day would be reduced.

Another option is to implement voting privacy security using elliptic curve cryptography. This approach reduces the computational requirements compared to 1024-bit integer arithmetic. The BBB-Voting white paper shows that a vote of up to 7 candidates can fit into the gas limit when using ECC Jacobi coordinates and optimizing scalar multiplication on an elliptic curve using the „Multiplication with Scalar Decomposition“ method, whereas a vote could have at most 2 candidates when using 1024-bit integer arithmetic. However, even such an implementation would not be conducive to allowing BBB-Voting to be currently deployed on Solana. After reducing the computational effort by $3.5\times$, the vote method of final prototype would still require $\sim 3.5\times$ higher compute budget than the current 200,000 compute units.

As for the price of voting, it is much lower on Solana than on Ethereum. The price of a transaction on Solana depends only on the number of signatures of a given transaction – 5000 lamports for each signature. The all time high price of Solana was around $200 by the time of this writing, and at that time the fee would have been $0.001. By contrast, the BBB-Voting white paper claims „Although we optimized the costs of our implementation, the expenses imposed by a smart contract platform might be still high, especially during peaks of the gas price“ – which in reality is units to tens of dollars per transaction (the price of Ethereum gas is very volatile). Ivana Stančíková even reports that in early May 2021, the cost to each voter would be $676 and $2,841 would be the cost to the authority [28]. The cost of creating a rent exempt account for voting from three candidates with 50 participants is 3,307,392 lamports, which during the all time high Solana token price translated to approx. $6.61. Moreover, such an account can be canceled at any later time and the amount can be refunded.

# Chapter 9

# Conclusion

The goal of this thesis was to analyze the suitability of the Solana blockchain for BBB-Voting and then to implement this voting system. Therefore, I familiarized myself with the structure and functioning of blockchain technologies and focused primarily on the Solana blockchain. In addition, I studied the principles of electronic voting and in especially the BBB-Voting protocol. I also got to know the Rust language and the framework for creating Solana smart contracts, Anchor.

Based on the knowledge gained, I created a proposal to implement a BBB-Voting system for Solana. I have consulted this proposal with one of the authors of the BBB-Voting system. Subsequently, I implemented the proposal and created the final prototype – a working voting system based on BBB-Voting running on the Solana blockchain. I also created tests to verify the functionality of the final prototype.

A number of problems were encountered during development, one of them fundamental to the actual deployment of the system on the Solana mainnet. This problem lies in the complexity of the implemented method for voting, within which the NIKZ proof is verified. This method consumes $\sim 19\times$ more computational units than Solana's current limit for a voting from 3 candidates. Therefore, I also had to create a modified version of Solana and use it for development.

However, it is possible that in the future it will be possible to deploy the solution on the Solana mainnet as Solana evolves and changes to the computational budget are foreseen. In that case, the switch to the Solana blockchain over the original Ethereum promises to significantly speed up elections – even a nationwide election in the US could be done in a few days. Also, the cost of voting is many times lower compared to Ethereum.

In addition, I have developed a front end for voting that allows the voting system to be used from a web browser. This is a single-page responsive web application developed using the ReactJS, React-Rexux, Material UI and Anchor web3 libraries.

I see opportunities for improvement in the area of privacy of voting protocol. To secure it against attacks, the system would need to be extended to use at least 1024-bit encryption keys, which is not possible natively in Rust as it only supports a maximum of 128-bit numbers. Another way to provide vote secrecy would be to use elliptic curve cryptography. In addition, the system could be extended to include recovery from erroneous states and scalability, which are extensions that exist for the BBB-Voting Ethereum version.

# Bibliography


[1] Adida, B. Helios: Web-based Open-Audit Voting. In: *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA* [[Online]]. Berkeley, California: USENIX Association, 2008, p. 335–348.

[2] Antonopoulos, A. M. *Mastering Bitcoin: Programming the Open Blockchain.* 2ndth ed. O'Reilly Media, Inc., 2017. ISBN 1491954388.

[3] Baudron, O. et al. Practical Multi-candidate Election System. In: *PODC '01.* New York, NY, USA: ACM, 2001, p. 274–283. ISBN 1-58113-383-9.

[4] Bell et al. STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. In: *EVT/WOTE 13* [[Online]]. Washington, D.C.: USENIX Association, 2013, p. 18–37.

[5] Cassez, F., Fuller, J. and Asgaonkar, A. *Formal Verification of the Ethereum 2.0 Beacon Chain.* 2021.

[6] Grincalaitis, M. *Mastering Ethereum: Implement advanced blockchain applications using Ethereum-supported tools, services, and protocols.* 1stth ed. Packt Publishing Ltd., 2019. ISBN 978-1-78953-137-4.

[7] Groth, J. Efficient Maximal Privacy in Boardroom Voting and Anonymous Broadcast. In: Juels, A., ed. *Financial Cryptography.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, p. 90–104. ISBN 978-3-540-27809-2.

[8] Hao, F., Ryan, P. Y. A. and Zielinski, P. Anonymous voting by two-round public discussion. *IET Information Security.* 2010, vol. 4, no. 2, p. 62–67.

[9] Haqshanas, R. *Solana Labs CEO Denies That Solana Went Down After a DDoS Attack.* 2022. Available at: https://cryptonews.com/news/solana-repotedly-went-down-again-after-ddos-attack.htm.

[10] Hayward, A. *Solana Blames 'Denial of Service Attack' for Last Week's Downtime.* 2021. Available at: https://decrypt.co/81375/solana-blames-denial-of-service-attack-for-last-weeks-downtime.

[11] Investments, G. *An Introduction to Solana.* 2021. Available at: https://grayscale.com/wp-content/uploads/2021/12/grayscale-building-blocks-solana-1.pdf.

[12] Joaquim, R. and Ribeiro, C. An Efficient and Highly Sound Voter Verification Technique and Its Implementation. In: *E-Voting and Identity.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, p. 104–121. ISBN 978-3-642-32747-6.

[13] Khader, D., Smyth, B., Ryan, P. Y. A. and Hao, F. A Fair and Robust Voting System by Broadcast. In: *(EVOTE 2012), July 11-14, 2012, Castle Hofen, Bregenz, Austria*. Bonn, Germany: GI, 2012, p. 285–299.

[14] Kiayias, A. and Yung, M. Self-tallying Elections and Perfect Ballot Secrecy. In: Naccache, D. and Paillier, P., ed. *Public Key Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, p. 141–158. ISBN 978-3-540-45664-3.

[15] Killer, C., Rodrigues, B., Scheid, E. J., Franco, M., Eck, M. et al. Provotum: A Blockchain-based and End-to-end Verifiable Remote Electronic Voting System. In: *2020 IEEE 45th Conference on Local Computer Networks (LCN)*. 2020, p. 172–183. DOI: 10.1109/LCN48667.2020.9314815.

[16] Kilmann, C. *How to mine Ethereum: A step-by-step guide*. 2022. Available at: https://www.businessinsider.com/personal-finance/how-to-mine-ethereum.

[17] Kuperberg, M. *Enabling Deletion in Append-Only Blockchains (Short Summary / Work in Progress)*. 2020.

[18] Lantz, L. and Cawrey, D. *Mastering Blockchain: A deep dive into distributed ledgers, consensus protocols, smart contracts, DApps, cryptocurrencies, Ethereum, and more*. 3rdth ed. Packt Publishing Ltd, 2020. ISBN 978-1-83921-319-9.

[19] Lee, M. *Solana: Scalability through speed*. 2022. Available at: https://www.nansen.ai/research/solana-scalability-through-speed.

[20] Li, Y., Susilo, W., Yang, G., Yu, Y., Liu, D. et al. A Blockchain-based Self-tallying Voting Protocol in Decentralized IoT. *IEEE Transactions on Dependable and Secure Computing*. 2020, p. 1–1. DOI: 10.1109/TDSC.2020.2979856.

[21] Matile, R., Rodrigues, B., Scheid, E. J. and Stiller, B. CaIV: Cast-as-Intended Verifiability in Blockchain-based Voting. In: *ICBC 2019, Seoul, Korea (South), May 14-17, 2019*. Washington, DC: IEEE, 2019, p. 24–28.

[22] McCorry, P., Shahandashti, S. F. and Hao, F. 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*. Berlin, Heidelberg: Springer-Verlag, 2017, p. 357–375.

[23] Park, S., Specter, M., Narula, N. and Rivest, R. L. *Going from bad to worse: from internet voting to blockchain voting*. DRAFT, 2020.

[24] Sandler, D., Derr, K. and Wallach, D. S. VoteBox: A Tamper-evident, Verifiable Electronic Voting System. In: *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA* [[Online]]. Berkeley, California: USENIX Association, 2008, p. 349–364.

[25] Seifelnasr, M., Galal, H. S. and Youssef, A. M. Scalable Open-Vote Network on Ethereum. In: Bernhard, M., Bracciali, A., Camp, L. J., Matsuo, S., Maurushat, A. et al., ed. *Financial Cryptography and Data Security - FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers*. Springer, 2020, vol.

12063, p. 436–450. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-54455-3_31.

[26] Shadrac, R. *Solana Network Crosses 1000 Active Validators on Mainnet Beta.* 2021. Available at: https://coinfomania.com/solana-network-1000-validators-on-mainnet-beta/.

[27] Shahandashti, S. F. and Hao, F. DRE-ip: A Verifiable E-Voting Scheme Without Tallying Authorities. In: *Computer Security - ESORICS 2016, Proceedings, Part II.* Berlin, Heidelberg: Springer-Verlag, 2016, p. 223–240.

[28] Stančíková, I. *Škálovatelné hlasování s ochranou soukromí hlasů založené na blockchainu.* Brno, CZ, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/24095/.

[29] Venugopalan, S., Homoliak, I., Li, Z. and Szalachowski, P. BBB-Voting: 1-out-of-k Blockchain-Based Boardroom Voting. 2021.

[30] Wood, G. Ethereum: A secure decentralised generalised transaction ledger. 2014.

[31] Yaga, D., Mell, P., Roby, N. and Scarfone, K. Blockchain technology overview. National Institute of Standards and Technology. Oct 2018. DOI: 10.6028/nist.ir.8202. Available at: http://dx.doi.org/10.6028/NIST.IR.8202.

[32] Yakovenko, A. Solana: A new architecture for a high performance blockchain v0.8.13. 2017.

[33] Zhang, W. et al. A Privacy-Preserving Voting Protocol on Blockchain. In: *CLOUD 2018, San Francisco, CA, USA, July 2-7, 2018.* Washington, DC: IEEE, 2018, p. 401–408.