Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Student Democracy with Blockchain

Bachelor's Thesis

Kaan Sentürk

skaan@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Darya Melnyk, Tejaswi Nadahalli
Prof. Dr. Roger Wattenhofer

March 17, 2019

Acknowledgements

First I want to thank my supervisors Darya Melnyk and Tejaswi Nadahalli for the opportunity to work on this Bachelor's Thesis in the Distributed Computing Group at ETH Zurich. With their support and guidance during my work, I learned a lot regarding e-voting protocols and blockchain technology.

Abstract

With the increasing importance of elections, new innovations like blockchain technology are analyzed and evaluated for improving our existing voting processes. For this purpose we implemented three protocols with HyperLedger Fabric (HLF), a framework for creating custom blockchain networks.

In this thesis we first give a brief introduction into e-voting and blockchain technology. Then we dive into our adapted implementation of the Open Vote Network (OVNet), which is a decentralised and self-tallying voting protocol for small boardroom elections. An application of this protocol was already published as a smart contract on the Ethereum public blockchain and is now adapted to a permissioned blockchain.

In the second part we cover a voting protocol for national scale elections, where we build an improved application of the homomorphic voting system proposed in B.Colin's thesis [1], which used the blockchain only as an immutable database to store the hashed voting data. The basis for his work was the multi-authority election scheme by Cramer et al. [2]. We propose that we use the network's consensus mechanism to also leverage the execution of the protocol itself by defining transactions for each operation of the protocol.

Contents

Chapter 1

Introduction

Democracy is a system where people exercise power by voting. The strength of such a system depends heavily on the trust set into the election process. Citizens are present when filing out their ballot, but what happens with a ballot after the vote has been cast? Often citizens just assume that the voting process is safe and that everything works correctly in postal votings. To make sure that the voting process is correct, engineers all over the world have been publishing papers of new voting protocols that guarantee some degree of safety against corruption. Nevertheless, there is still no single best solution yet. A part of this problem is due to voting protocols solving different challenges and therefore having different requirements. Homomorhpic systems for example have the downside that a lot of non-interactive zero knowlege proofs are necessary to prove correctness, while e-voting protocols based on mix-net are centralized and depend on the election authorities.

And with the introduction of Bitcoin [3], there is now a new innovation to explore. An innovation that removes power from central authorities and distributes it among multiple peers. This leads to new protocols and new ways for improvement regarding distributed execution being discovered. A key interest of this thesis is finding and improving multiple protocols with this new blockchain technology such that we can make new assumptions about the usefulness of blockchains in e-voting.

In this thesis we focus on two protocols that already have their implementations and improve them by using blockchains. In the first part we adapt the Open Vote Network (OVNet), which was already designed for a public blockchain, and enhance its access control by using permissioned blockchains. In the second part we develop two similar protocols, that are thoroughly studied but do not have their application in a permissioned blockchain yet. The second part builds upon the work of Berner [1], who proposed a protocol using blockchain as an immutable hashed copy of voting data. We go a step further and integrate the whole execution of the protocol into the blockchain.

Chapter 2

Related work

A basis for the Open Vote Network (OVNet) was first proposed by by Hao et al. [4] called two-round anonymous veto protocol, which was significantly more efficient in terms of number of rounds and computational cost compared to related work. With the OVNet provided by [5], the protocol got integrated into the Ethereum public blockchain. With this thesis we improve this protocol regarding security and privacy by restricting access to the peer-to-peer network and by hiding transactions and private state variables, which would be publicly readable in public blockchains.

The implementation mentioned above is just one out of the several applications on Ethereum. To mention a few, there is PLCRVoting [6], which uses ERC20 tokens in their smart contracts to carry out the election, and there is PublicVotes [7], which is a publicly verifiable e-voting contract. Other implementation that use a similar tallying like the OVNet, but build upon other blockchain technologies are VotoSocial [8] and Follow My Vote [9], which also uses an ellyptic curve cryptography like the OVNet implementation from McCorry et al. [5] in difference to our solution. The WaveVote [10] application for example is even build on top of the OVNet.

The homomorphic e-voting protocol has already been used with different variations. An implementation with a public blockchain for example was provided by Polys [11], which also uses Shamir's secret sharing [12], but integrates the full protocol into the blockchain in difference to Berner's protocol. The difference to our work is the usage of permissioned blockchain to restrict peers in the network and shield transaction visibility. Another implementation is the platform build by Voatz [13], a startup that also used HyperLedger Fabric to carry out elections based on mix-net systems. There are multiple other applications for mix-net systems and applications without blockchains less related, which renders a detailed description of all those solutions out of scope for this related work section.

This thesis mainly focuses on improving two protocols with permissioned blockchains and therefore is strongly related to the OVNet implementation and the Homomorphic E-Voting proposed by Berner [1] with the basis of Cramer et al. [2].

Chapter 3

Preliminaries

3.1 E-Voting

E-voting protocols need to solve some difficult challenges, but will essentially cover an important role in future elections. For this purpose, the e-voting protocols need to fulfil multiple requirements and properties. We explain some of the key requirements a voting system needs to cover and some of the less crucial requirements that we also tried to improve during the implementation of our applications. This list represents only properties that we have focused on and should not be considered complete.

Authenticity Only voters that are eligible to vote can cast their votes. Multiple votes from a voter and a change of an already cast vote is prohibited. This requirement is fulfilled with access control of participants and protocol implementation.

Correctness The result of the tally represents the result that the voters voted for. All cast votes are counted and no single vote has been modified after being cast. This requirement is fulfilled with immutability of processed data, queries and transactions that fetch all cast votes and tally the result again.

Privacy The cast votes of all voters are unknown for authorities and the remaining voters. This requirement is fulfilled with encryption and permissioned encapsulation of voting data.

Auditing Voters can verify the result of a voting and verify the correct execution of the protocol. Voters therefore are able to verify the correct handling of cast votes. This requirement is fulfilled with zero knowledge proofs and queries of the voting data.

Decentralization Through a decentralized protocol the failure of a system by a single authority is prevented. The execution of the protocol is handled by multiple authorities and voters. This requirement is fulfilled with the

usage of blockchain technology that distributes the trust and execution of the protocol to multiple peers in a network.

Robustness The system is robust against corruption and external attackers. The fulfillment of this requirement is improved with access control for participation in the network and parameter and error handling of issued transactions.

Simplicity The simplicity covers the understanding of the implementation and user experience of the election system. The fulfillment of this requirement is improved with reducing the necessary interactions for voters and decoupling different phases of the protocol into smaller and simpler phases.

For a deeper introduction into these requirements the Semester thesis of B.Collin [1] and the requirement set for the protocol specifications in CHVote [14] are good starting points.

3.2 Blockchain

We define the properties of a blockchain and describe some important concepts to justify our intention of using HyperLedger Fabric to develop these applications.

A blockchain in its simplest form can be seen as a decentralized immutable database, managed by a peer-to-peer network. The nodes in the network agree on the state of the database through distributed consensus mechanisms. The blockchain is built by coupling multiple transactions into a block and then connecting new blocks of transactions in a sequence or chain. Starting at an initial block, an immutable (append-only) history of transactions is maintained.

The first published blockchain was Bitcoin [3], which is a decentralized financial ledger, whose main purpose was to replace the role of a central authority in maintaining financial ledgers by storing transactions. Transactions in Bitcoin are transfers of value between Bitcoin addresses. A user that owns such an address, has a private key, which is used for signing new transactions with a signature and in this way providing a cryptographic proof of the ownership of the address. The ledger keeps track of all transactions ever issued and allows network clients to validate each transaction and ensuring that the spent balance is actually owned by the sender address.

Each block in the Bitcoin blockchain contains in its header a timestamp, a block number, transaction data, a cryptographic hash depending on the blocks content and a reference to the previous block hash located in Figure 3.1 inside the meta data container. Multiple blocks are chained together with references to previous blocks. By following the transactions in the blockchain, users can calculate the balance possessed by a specific address. Through the cryptographic

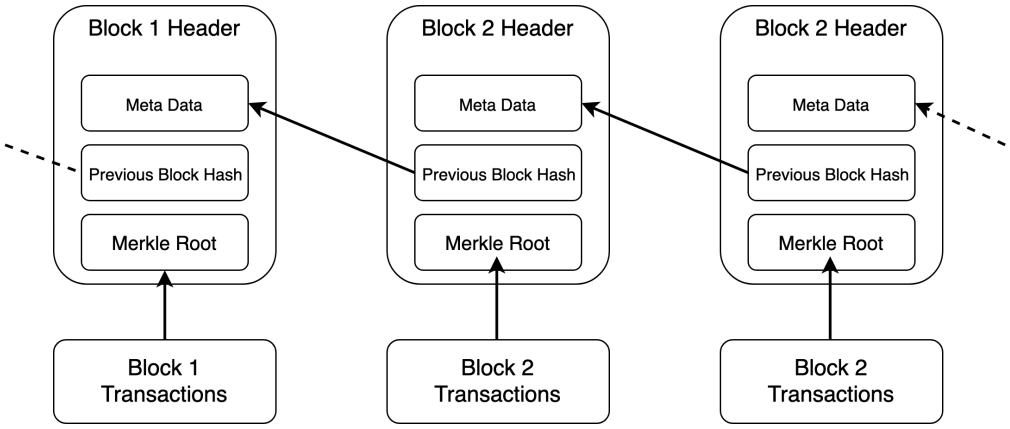

Figure 3.1: Simplified model of a blockchain with blocks and its contents

linking of blocks with block hashs, a blockchain is resistant to modifications of historical data. A change in the content of a previous block that is already minded and processed, would lead to a new block hash in the previous block and therefore invalidate reference to the previous block in the current block.

3.3 Transaction Processing

In general a transaction represents a change of assets. A user issues a transaction that is signed with the private key of the address and then put into a transaction pool. Miners then confirm the transactions and include them in the block chain by solving a difficult problem in the distributed consensus algorithms (e.g. Proof of Work, Proof of Stake, ...). To process this transaction, a fraction of the transferred amount is marked as reward for processing to create an incentive for miners. As soon as a miner solves the task for a block, the created block with a proof of its validity is published among the peers. Since multiple miners can solve the problem at the same time for the next block, different blocks may get appended to the locally stored blockchain. Normally the first peer solving the task will have more time to distribute its block among the peers. This leads to inconsistency on the state of the blockchain between the peers in the network, which is solved by accepting the chain with the longest sequence of blocks. Transactions not being processed by the accepted blocks are placed back to the transaction pool.

It is important here to note that existing data can be modified by using a new transaction, but past transactions cannot be modified. This immutability feature of the blockchain technology could already enhance e-voting protocols by storing voting data that should not be modified. Nevertheless in this state the infrastructure still lacks important requirements. Because Bitcoin only supports

transactions that transfer asset value, it is hard to abstract the voting data with balances and asset value. On top of that, other requirements like authenticity, privacy, and verifiability are also neglected.

3.4 Smart Contracts and Chain Code

Fortunately, the innovation with blockchain did not stop with cryptocurrencies. A few new architectures were build that supported not only transfers of assets but also enabled the execution of code. Ethereum for example extended its functionality with a concept called smart contracts [15]. A smart contract is like a regular address with a balance. In addition, smart contracts also have storage for state variables and executable code (Chain Code). Users can invoke chain code, which then can update state variables or issue new transactions on behalf of the smart contract.

With this feature of a blockchain, we are already able to solve more problems regarding e-voting protocols including the key requirements of authenticity, privacy and verifiability. Since chain code is written in Turing-complete language, we can execute the protocol by issuing transactions and invoking chain code. The data structure for voting data is not restricted to cryptocurrencies anymore, since state variables for different parameters of the voting protocol can be defined.

3.5 Public blockhains vs. Permissioned blockchains

In a public blockchain anyone with the correct software can participate in the network. This makes sense, since currencies aim to be universal. The problem is that for some applications including e-voting, a public blockchain comes with a few limitations:

Security: Allowing everyone access to the network exposes additional risk. Vulnerabilities can be exploited easily, since the chain code of smart contracts is publicly available. For particular use cases, it is necessary to allow only eligible voters to participate in the network.

Privacy: Public blockchains are transparent and transactions are visible for anyone in the network. This can lead to knowledge about certain actions of voters like the participation in a particular voting.

Poor performance and Scalability: Consensus mechanisms for public blockchains are very expensive, since all transactions are processed on all nodes in the network. The throughput is low and the scalability is very restricted. In a country with millions of voters, the validation of transaction in a national scale election could already be impossible.

Cost: To process pending transactions an incentive system for miners is created in public blockchains such that a transaction fee is due on correct mining of a block. This can lead to high fees on applications with high transaction volumes like in e-voting protocols.

Because of these limitations, companies have developed new blockchain architectures that trade openness for privacy and security and in this way leverage the usage of blockchain technology for custom applications. One of these architectures are permissioned blockchains, where nodes need permission to participate in the network. Transactions and assets have access control for CRUD operations. This leads to participants being authenticated parties. For some e-voting protocols this makes sense, since voters are verified if they are eligible to participate in an election. Therefore the security and privacy of such blockchain applications is increased with permissioned blockchains.

Another advantage of permissioned blockchains is that different consensus protocols can be used due to the number of nodes being limited and known. Protocols like Practical Byzantine Fault Tolerance [16] can be used to increase performance and reduce cost. Furthermore, incentivized miners are not needed to maintain the network, since voting authorities and voters that build a consortium have an interest in executing the voting protocol and therefore making cryptocurrencies and transaction fees inutile.

3.6 HyperLedger Fabric

HyperLedger Fabric (HLF) is one of the best known frameworks for creating such custom permissioned blockchains. The modular architecture of HLF provides developers a way to customize consensus mechanisms and membership services. We can customize the architecture of the blockchain to our needs and handle access control for participants in our network very efficiently. Another strength of HLF is that developers can define their own transactions and specify the visibility of those transactions in different channels for different participants connecting to peers of the network.

The organizations in the consortium that build the HLF network are called members and those members setup their peers to participate in the network. Peers are identities that are created with a HLF component called Certificate Authority that creates cryptographic signatures. Peers inside a member organization are organized in channels and receive transaction requests from clients inside this channel. Those requests are then delegated to other peers and distributed in the network. All peers maintain one ledger per subscribed channel and handle different functions in the consensus mechanism of the distributed ledger technology (DLT). Unlike public blockchains, the members can setup different

type of peers that have different roles. The possible peer types and their main roles are listed below:

Endorser peer:

- receives and validates transaction requests from clients
- executes chain code and simulates result of transaction without updating the ledger

Anchor peer

- receives updates from orderer peers
- distributes updates to all other peers

Orderer peer

- central communication channel for network
- validates result of chain code execution and updates ledger in a channel
- distributes updates

Altough it would be possible to scale the network and create multiple members with multiple peers and different roles, we did not focus on the scalability but on the proof of concept of the voting protocols. In a real life scenario we could build a network, where we setup endorser peers and anchor peers for every local community and combine multiple local communities in a region with channels. One Orderer peer per regional communities could then be assigned for distributing updates in a national scale. Other possibilities of building good networks and a detailed explanation of all components can be found in the introduction to HyperLedger Fabric docs [17].

Chapter 4

Open Vote Network

The Open Vote Network [5] is a decentralized and self-tallying voting protocol. It is suitable for smaller boardroom elections and has been developed such that the protocol execution can be implemented in smart contracts. In general, the participants of a boardroom election are publicly known and the scale is limited. The initial implementation was published as smart contract for Ethereum [15]. The main strength of the Open Vote Network lies in the independence of a single authority when calculating the tally of the election, since the tally is decrypted by the voters themselves. Instead of trusting a single server architecture to provide the correct protocol for the election, it relies on the distributed consensus mechanism of the Ethereum blockchain. Due to public blockchains having limitations like security, privacy and cost like described in the preliminaries, we decided to use HyperLedger Fabric (HLF) to create a custom permissioned blockchain.

4.1 Preliminaries

In this section, we assume that for two primes p, q with $p = 2q - 1$, a finite cyclic group G_p of prime order p and a generator g are given.

4.1.1 ElGamal encryption

The ElGamal encryption [18] is a public-key encryption that builds upon the Diffie-Hellman key exchange [19, 20]. The ElGamal encryption contains three steps.

Key encryption We choose a random private key $s \in \{1, \dots q\}$. The public key is then determined with the generator and published publicly.

$$h = g^s$$

Encryption The encrypted ciphertext c of a message m consists of the pair (x, y). The message m must be an element of the finite cyclic group G_p. First, the sender chooses a random value $r \in [0, q-1]$ and determines the encrypted parameters.

- $x = g^r \mod p$
- $y = m \cdot h^r \mod p$

Resulting in the ciphertext $c = (x, y)$

Decryption For the decryption we then can determine the message m with the private key s as follows

$$m = \frac{y}{x^s} \mod p$$

The correctness of the equation follows by substituting

$$y = m \cdot h^r \mod p = m \cdot (g^s)^r \mod p$$

$$m = m \cdot g^{sr} \cdot (g^{rs})^{-1} \mod p$$

4.1.2 Schnorr proof

The Schnorr proof [21] is a proof of knowledge for discrete logarithms in a finite cyclic group G_p with a generator g. The proof can be made non-interactive via the Fiat-Shamir heuristic.

Protocol To prove the knowledge of $s = \log_g h$ the prover executes following protocol:

1. the prover chooses random value $r \in G_p$ and sends $t = g^r$ to the verifier
2. the verifier replies with a random challenge $c \in G_p$
3. the prover calculates $x = r + cs$ and sends the response x to the verifier

The verifier validates the proof if $g^x = th^c$

Fiat-Shamir Heuristic The Fiat-Shamir heuristic [22] is a technique for making interactive zero knowledge proofs non-interactive by choosing a cryptographic hash for the challenge in proof of knowledge. The Schnorr proof gets altered to following protocol to make it non-interactive:

To prove the knowledge of $s = \log_g h$ the prover executes following protocol:

1. the prover chooses random value $r \in G_p$ and sends $t = g^r$ to the verifier
2. the prover chooses $c = H(g, h, t)$, where $H()$ is a hash function
3. the prover calculates $x = r + cs$ and sends the response x to the verifier

The verifier validates the proof if $g^x = th^c$

4.2 Protocol

The Open Vote Network is a two-round protocol for voters with an optional third round for vote commitment. In the first round, all voters sign up for a particular voting instance and present their intention to participate in the election. In the second round, the voters broadcast their votes. With the self-tallying property of the protocol, every voter is able to tally the result of the election by themselves.

Setup For the election a finite cyclic group G_p of prime order p and a generator $g \in G_p$ is specified by an authority or distributed by n voters. A list of eligible voters $(P_1, P_2, \ldots, P_n)$ is established and each voter P_i selects a private key $x_i \in_R \mathbb{Z}_q$. The $ZKP(x_i)$ is a Schnorr proof made non-interactive with thie Fiat-Shamir heuristic like explained in the preliminaries.

Round 1 / Sign up Every voter P_i broadcasts his voting key g^{x_i} to the bulletin board together with a zero-knowledge-proof $ZKP(x_i)$ to prove the knowledge of a secret key. At the end of the sign up phase all voters validate the zero knowledge proofs of the other voters and compute a list of reconstruction keys:

$$Y_i = \prod_{j=1}^{i-1} g^{x_j} / \prod_{i+1}^{n} g^{x_j}$$

By setting $Y_i = g^{y_i}$, we can ensure that $\sum_i x_i y_i = 0$.

Round 2 / Voting: Every voter broadcast their vote $g^{x_i y_i} g^{v_i}$, where $v_i \in \{0, 1\}$, and a one-out-of-two zero knowledge proof to show a valid vote v_i with the Cramer, Damgard and Schoenmakers (CDS) [23] technique.

Tallying: All zero knowledge proofs are then verified and the tally is computed homomorphically with $\prod_i g^{x_i y_i} g^{v_i}$. Since $\prod_i g^{x_i y_i} = 1$, we can calculate the result with the discrete logarithm of $g^{\sum_i v_i}$. The discrete logarithm is bounded by the number of voters and thus easily solvable.

For the tally to be computed correctly and the election to succeed, it is necessary that all voters who sign up for the election, cast a vote in the voting phase. Note that the last voter who casts his vote could calculate the tally and its result before publishing his vote. This problem is tackled with an additional round, where voters commit to their vote.

4.3 Implementation

To adapt the Open Vote Network to the HyperLedger Fabric framework, we implemented the protocol by defining the assets that are stored in the distributed ledger and the transactions that change these assets.

4.3.1 Assets

In a financial ledger we have a transfer as an asset and the transfered amount as a field in the asset. Similarly we create assets for a voting system by defining the asset model for the permissioned blockchain.

Participant The participant asset defines a user in our network that is connected to a real identity. We differentiate between an AdminParticipant, who participates as an administrator and therefore can create a voting with eligible voters, and a VoterParticipant, who can be an eligible voter for some voting asset instance. Only the identity that has the correct access rights can see and interact with its participant instance.

participantKey: key of participant instance

Voting The Voting asset is one instance of an election. Only AdminParticipants can create a voting asset. Read and update is enabled also for VoterParticipants, since voters can calculate the tally by themselves.

votingKey: key of voting instance

state: current phase of the election

whitelist: array of eligible voters

order: order p of the finite cylcig group G_p

gen: generator g in G_p

tally: homomorphic tally of the votes

Voter The Voter asset is one instance connecting a participant to a specific voting instance. In this way we have a platform that enables the creation of multiple votings by allowing a participant to have one voter instance per voting if eligible. The voter asset is publicly readable, but creation and update is restricted to the identity with the correct assess rights.

voterKey: key of voter instance

publicKey: public key of voter for encrypted messaging

hash: challenge used in $ZKP(x_i)$

sig: signature used in $ZKP(x_i)$

reconstructedKey: key for decryption of tally

vote: encrypted vote

hashedVote: commitment to vote

voting: reference to voting asset

participant: reference to participant asset

4.3.2 Transactions

For the transactions that we want to create, we have a specific flow for the voting protocol.

CreateVoting: The CreateVoting transaction is straightforward, since the parameters given with the transaction call are just stored in a new voting asset. Only AdminParticipant can issue this transaction.

EnableVoter: An additional transaction for adding eligible voters if not already added during the CreateVoting transaction. Only AdminParticipant can issue this transaction.

StartSignup: The StartSignup transaction can be issued to update the state of a voting asset. Before this transaction is issued, the CreateVoter transaction cannot be issued. Only AdminParticipant can issue this transaction.

CreateVoter: The CreateVoter transaction checks whether the voting asset is in the SIGNUP state and creates the voter asset with references to the participant and the voting asset. Only VoterParticipant can issue this transaction.

StartVoting The StartSignup transaction can be issued to update the state of a voting asset. In this transaction all voters private key is checked for validity with Schnorr proof explained in the preliminaries section. Before this transaction is issued, the Vote transaction cannot be issued. Only AdminParticipant can issue this transaction.

Vote The Vote transaction checks whether the voting asset is in the VOTING state and updates the voter asset of the participant regarding the specified voting. Only VoterParticipant can issue this transaction.

StartTallying The StartTallying transaction checks whether all registered voters have broadcasted their vote and all votes are valid. AdminParticipants and VoterParticipants can issue this transaction.

There are some additional transactions implemented, two for checking the two zero knowledge proofs for valid secret keys and valid votes, and one transaction for computing the tally from the broadcasted votes.

4.4 Conclusion

With the Open Vote Network adapted to HyperLedger Fabric, we provide an implementation that increases security and privacy through explicit handling of access control in the blockchain architecture. Additionaly, a permissioned blockchain lacks the necessity of cryptocurrency payments for transactions or execution of code, since miners do not need to be to incentivized. Members in the consortium that have an interest in the execution of the protocol handle the mining of the transaction by themselves.

The problem with OVNet is the limitation on the number of voters and scalability issues due to necessity that every voter needs to participate in the decryption phase. This aspect could be improved in a future work. Another point that could be improved upon are the key parameters used in the protocol. Currently the standard variable types with 64 bit are used for numbers like the prime order or the generator. Extending these variables and the transactions to support array types would increase the security of the protocol greatly. The focus lied here mostly on access control and leverage of privacy and usability.

Chapter 5

Homomorphic Voting

The OVNet is limited in its scalability and therefore only suitable for boardroom elections. To improve and provide different protocols using blockchains, we want to introduce an alternative implementation building upon Berner's thesis [1].

The protocol proposed by Berner is a voting system with distributed homomorphic encryption and threshold decryption. This allows removing trust from a single authority while at the same time keeping the protocol scalable regarding the number of voters. The initial implementation only used the blockchain as an immutable database to store the hashed voting data. We propose that the blockchains consensus mechanism can also enforce the execution of the whole protocol. For this purpose we filtered out the main components of the protocol and implemented two versions that differ in the key generation for the election system.

In the first voting system we have a Shamir's secret sharing [12], where an independent authority generates a random polynom of grade t, a secret key s and a public key h for the election. This version uses a different key generation mechanism than the work of Berner. The protocol is faster since there is no need for distributed generation of shares. The disadvantage is the trust set in a single authority and the fact that the private key is known to an authority.

In the second improved version we use Pedersen's distributed key generation [24, 25] for generating a random polynom of grade t and shares for the decryption. In this way the private key is never generated.

5.1 Preliminaries

For this preliminaries section we assume that a finite cyclic group G_p of prime order p satisfying $p = 2q - 1$ for another prime q , a generator g, a secret key $s \in [0, q-1]$ and numbers n, t with $t \leq n$ is given. The generation of this parameters is explained later in the protocol section.

5.1.1 Shamir's Secret Sharing

The basic idea of Shamir's Secret sharing [12] is that each point of a polynom of grade $t-1$ can be reconstructed with t random points on this polynom. The secret key can then be generated at position $x = 0$ of the polynom. To determine secure parameters for the encryption system, the values must be chosen such that the decisional Diffie-Hellman (DDH) [19, 20] assumption is satisfied. To generating a secret key we proceed as follows:

1. We choose a random polynom $s(x)$ of grade $t-1$, where t denotes the threshold for the minimum number of subkeys necessary to determine the private key
$$s(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1} \mod q, \quad 0 \leq a_i < p$$
2. The private key s equals $s(0)$ and is kept secretly. The public key is published to the blockchain.
$$h = g^s \mod p$$
3. For all n decryptors a value x_i with $i \in [1..n]$ is randomly selected and a pair $(x_i, s(x_i))$ is created. While x_i is published publicly, the share $s(x_i)$ of the i-th decryptor is encrypted.

To determine the private key from the shares, we construct the polynom with lagrange interpolation at $x = 0$. The lagrange polynom is calculated with the x_i values publicly available for each decryptor:

$$\lambda_i(0) = \prod_{j=1, j \neq i}^{k} \frac{-x_j}{x_i - x_j} \mod q$$

In this equation the product is taken from k decryptors, which needs to be at least the threshold t to determine the correct private key:

$$s(0) = \sum_{i=0}^{k} s(x_i)\lambda_i(0) \mod q$$

5.1.2 Pedersen Distributed Key Generation

The Pedersen DKG [24, 25] is a distributed version of Shamir's Secret Sharing [12], where each decryptor executes a Shamir's Secret Sharing. Every decryptor creates a polynom and shares for the other decryptors.

1. Every decryptor D_i receives a value x_i publicly on the bulletin board with $i \in [1..n]$ and every decryptor D_i selects a random polynom of degree $t-1$.

$$s_i(x) = a_{i0} + a_{i1}x + a_{i2}x^2 + \cdots + a_{it-1}x^{t-1} \mod p$$

The coefficients a_{ik} are kept secret and every decryptor D_i sends the value $s_i(x_j)$ with $j = 1, \ldots, n$ encrypted with the signature to the decryptor D_j.

2. Every decryptor D_i evaluates $h_i(x)$

$$h_i(x) = g^{s_i(x)} \mod p = g^{a_{i0}} \cdot g^{a_{i1} * x} \cdot \cdots \cdot g^{a_{it-1}x^{t-1}} \mod p$$

The coefficients h_{it} of $h_i(x)$ are then published to the public bulletin board, where $h_{i0} = g^{a_{i0}}, h_{i1} = g^{a_{i1}}, \ldots$

3. Every decryptor D_j validates the h_{it} with the received shares $s_i(x_j)$ and reports decryptors with invalid provided shares.

$$h_i(x_j) = g^{s_i(x_j)} = \prod_{t=0}^{t-1} (h_{it})^{x_j^t}$$

If a corrupt decryptor D_i is reported from multiple decryptors, the parameters are reinitialized such that $h_{it} = 1$, $a_{it} = 0$ for $t \in [1..(t-1)]$ and $s_i(x_j) = 0$ for $j \in [1..n]$

4. The decryptors D_i determine their private share.

$$s_i = \sum_{j=1}^{n} s_j(x_i) \mod p$$

And every decryptor validates the public key.

$$h = \prod_{j=1}^{n} h_{j0} \mod p$$

From this part the same last two steps like in the Shamir's Secret Sharing can be executed to calculate the lagrange interpolation for the position $x = 0$. The disadvantage in this case is that the private key will get generated. Another option is to prevent this by executing a different protocol and generate only partial decryption. A protocol that does this is explained next.

5.1.3 Partial threshold decryption

We assume to have a ElGamal encrypted message $tc(tx, ty)$

1. Every decryptor D_i takes his share s_i and generates a subkey w_i

$$w_i = tx^{s_i} \mod p$$

With this subkeys we can reconstruct the message without determining the private key by solving

$$tx^s = \prod_{j=1}^{k} w_i^{\lambda_i} \mod p$$

$$tm = ty \cdot tx^s \mod p$$

In this way tm is the decryption of tc and the private key s has never been determined.

5.2 Protocol

1. Decision of election parameters: The authorities decide on a finite cyclic group G_p of prime order p, such that $p = 2q - 1$ for another prime q. A generator g for a subgroup of G_p is created such that g is a quadratic residue of G_p, meaning for a random integer x, the generator g satisfies:

$$x^2 \equiv g \mod n$$

In this way the generator g, creates a subgroup of G_p that only contains quadratic residues. Meaning that with the information of $g^r \mod p$ being a quadratic residue or not, there is no additional information gained for the value of $g^{sr} \mod p$ and the subgroup contains at least q elements. The secret key s has to be chosen in the interval $s \in [0, q - 1]$.

2. Key generation: In this step of the process our two versions differ. In our first implementation we use the Shamir's Secret sharing, where a single admin authority creates a secret key and distributes the shares to the decryption authorities. The distribution of the shares needs to be private between the authority and the decryptors, for this the shares are encrypted via ElGamal encryption. The downside to this protocol is that the private key is known by the admin authority and even is determined by the decryption authorities when decrypting the tally.

In the second implementation we improve this aspect and determine the private key of the election with Pedersen's distributed key generation protocol explained in the preliminaries. The advantage is that with the partial threshhold decryption also explained in the preliminaries, the private key is never fully determined, since we use partial subkeys to decrypt the tally of the election.

The result of both protocols is a secret key s and public key for the election system that can be used to cast votes, where

$$h = g^s$$

3. Voting and Tallying: When the setup for the election system is done, voters V_i can cast their votes encrypted with ElGamal encryption system $c_i = (x_i, y_i)$ and the public key of the election. In order to tally encrypted ciphertexts, the messages that represent the votes need to have a special form. For this purpose we choose a value m_0 from the finite cyclic group G_p that represents the binary value 0. The inverse $m_1 = m_0^-1$ then represents the binary value 1 for elections with two vote options.

The tally of all votes can then be computed homomorphically without decrypting the votes beforehand

$$tx = \prod_{i=1}^{n} x_i \mod p \qquad ty = \prod_{i=1}^{n} y_i \mod p$$

4. Decryption: After the tally has been computed we have an ElGamal encrypted message $tc(tx, ty)$.

With a zero knowledge proof the decrypting authorities proof that the used share is valid. For the proof a Schnorr proof made non-interactive with the Fiat-Shamir heuristic can be used.

$$\log_g g^{s_i} = s_i = \log_{tx} w_i$$

If all shares are valid the encrypted tally can be decrypted by reconstructing the private key with Lagrange interpolation like explained in the preliminaries with Shamir's secret sharing or with partial threshold decryption to determine the decrypted tally without calculating the private key with Pedersen's distributed key generation. This leads to a decrypted message $tm = m1^{y-n}$, where y is the number of yes (binary 1) and n is the number of no (binary 0) votes.

To receive the tally out of this message the value $y - n$) can be determined via the discrete logarithmus, which is a hard problem. But since the number of voters is relatively small, we can calculated the discrete logarithm for example via baby-step giant-step algorithm [26].

5.3 Implementation

To adapt the protocol to HyperLedger Fabric, we take a similar approach like in the Open Vote Network example. For this we first define what assets the ledger maintains. In a second step we focus on the defined transactions for the blockchain, their implementation and how they update the assets.

5.3.1 Assets

We have three different asset types that interact with each other. First there are the participant assets, which are actors inside the business network application that represent the outside identities. Then we have the voting assets that represent the election system split up in a publicly available asset called PublicVoting and a private asset called SecretVoting, which is only available for specific AdminParticipants. As last asset type there is the voter asset types, which is also split up in a publicy available asset called PublicVoter and a private assets called SecretVoter that are only available for VoterParticipants with the correct access rights.

AdminParticipant: The AdminParticipant is an administrator/authority for the election system. With the role of an administrator we can create voting assets and new elections.

participantKey: key of participant instance

VoterParticipant: The VoterParticipant is a participant inside the network that represents an outside voter. Every voter is an AdminParticipant and can therefore create its own elections, since the VoterParticipant inherits from AdminParticipant.

participantKey: key of participant instance

PublicVoting: An instance of the PublicVoting asset is always created together with an instance of the SecretVoting asset. It represents one specific election and contains all necessary parameters for the successful execution of the protocol. Everyone can create an instance, but only the creator of the instance is able to update the particular PublicVoting and SecretVoting instance.

The necessary parameters are a unique identifier for referencing particular instances, a state enumeration parameter that indicates the current phase of the election, decryption parameters that are necessary for the decryption phase, execution parameters like the prime order p of the finite cyclic group G, the generator g and the threshhold t, and result parameters for the tally.

Depending on the key generation process (Shamir or Pedersen), different access control permissions for updates of the asset instance is established.

votingKey: key of voting instance

state: current phase of the election

whitelist: array of eligible voters

decryptors: array of participating decryptors

points: points for lagrange interpolation of the decryptors

votes: array of valid votes $m_0, m_1, \ldots$

order: order of the finite cylcig group G

sub: order q of the subgrouf of G_p

gen: generator g in G

threshold number of decryptors needed for decryption

publicKey: public key of the election for encrypting votes

tallyX, tallyY: homomorphically tallied ElGamal encrypted votes

SecretVoting: The SecretVoting asset contains further data for the election system that need to be kept private and hidden from voters and other authorities. Here we store the coefficients of the randomly generated polynom for the e-voting protocol. Depending of the key generation (single or distributed), the shares are also stored in this asset.

A single instance of an election is divided into two instances of the PublicVoting asset and the SecretVoting asset, because at the time of development of this application, there was no possibility to specify different access permissions on different attributes of an instance.

votingKey: key of voting instance

secretKey: secret key of the election

shares: shares of the private key

coefficients: coefficients of the generated random polynom

The SecretVoting asset is empty if the private key, the random polynom and the shares are generated distributed via Pedersen's DKG. The fields are therefore declared optional. In the version with Pedersen's DKG, the SecretVoting asset is ignored.

PublicVoter: An instance of the PublicVoter asset is created together with an instance of the SecretVoter asset on sign up of a VoterParticipant to a particular election. In this way our application can have multiple elections, since we generate for each election a new voter pair referencing a particular voting for a VoterParticipant. The PublicVoter asset defines publicly available attributes of a voter, encrypted parameters for shares and votes and parameters necessary for zero knowledge proofs for the voter's secret key and for the validity of the vote.

Note that (*messageX, messageY*) is the ElGamal encrypted share of the voter on the election's private key and that (*voteX, voteY*) is the ElGamal encrypted vote of the voter. The (*voteX, voteY*) for all voters is homomorphically tallied in the tallying phase and the result is then stored in the corresponding instance of the PublicVoting asset.

voterKey: key of voter instance

publicKey: public key of voter for message encryption

hash: challenge used in $ZKP(x_i)$

sig: signature used in $ZKP(x_i)$

shareHash: challenge used in $ZKP(s_i)$

sigHash: signature used in $ZKP(s_i)$

point: interpolation point for share s_i

messageX, messageY: encrypted share distributed from authorities

voteX, voteY: encrypted vote

decryptor: published part of key when participating in decryption

lagrange: published lagrange coefficient when participating in decryption

voting: reference to voting asset

participant: reference to participant asset

SecretVoter: The SecretVoter asset contains further data for a voter in a particular election that need to be kept private and hidden from the other participants in the network. Here we store the secret key of the voter and the share on the private key of the election used in the decryption phase if the voter wants to take part in the decryption.

voterKey: key of voter instance

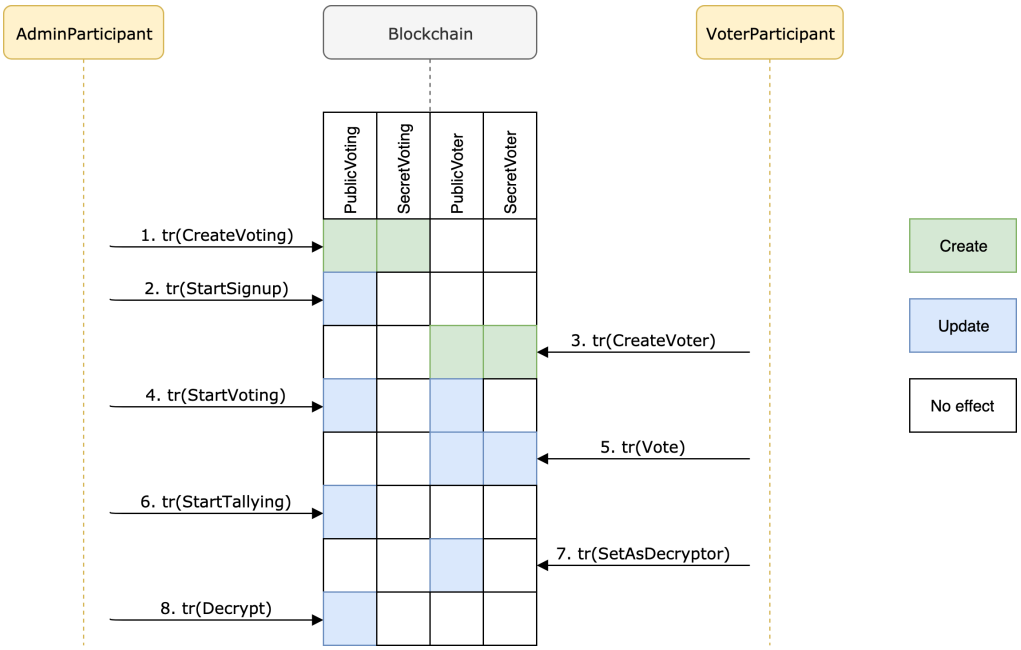

Figure 5.1: Timeline of transactions and effect on assets

secretKey: secret key of voter for private messaging

share: share of private key

voting: reference to voting asset

participant: reference to participant asset

5.3.2 Transactions

For the transactions we can see in Figure 5.1 that AdminParticipants and VoterParticipants interact with the blockchain and execute the voting protocol by issuing new transactions. Those transactions either create new instances of a specific asset, update an existing instance of an asset or have no effect on the asset.

Note that in the figure the rows of the table do not represent blocks of the blockchain but the effect of a transaction on the assets. Multiple transactions could be mined together depending on the data payload of the transactions in the block.

CreateVoting: The CreateVoting transaction can only be issued by an AdminParticipant and is the initial transaction to start an election. Depending

on the application either a Shamir's Secret Sharing (SSS) or Pedersen's DKG (PDKG) with multiple authorities is executed after checking the validity of the parameters. The shares s_i of the private key s. The transaction then creates new PublicVoting and SecretVoting instances with the created election parameters.

StartSignup: The StartSignup transaction can only be issued by an AdminParticipant and comes like in Figure 5.1 after CreateVoting and before CreateVoter. The only purpose of this transaction is to move the election system to the next phase and allow VoterParticipants to sign up to the particular voting referenced with the *votingKey* parameter.

CreateVoter: The CreateVoter transaction can only be issued by a VoterParticipant if the voting instances are in the correct phase. The CreateVoter transaction creates two instances, one for the PublicVoter asset and one for the SecretVoter asset. Multiple parameters for voter signature and zero knowledge proofs are generated and stored in the blockchain.

StartVoting: The StartVoting transaction is issued by AdminParticipants. First the private keys of the signed up voters is tested for its validity with Schnorr's zero knowledge proof. Next the shares are encrypted and distributed among the voters. If all proofs are valid and the distribution of the shares is handled successfully, the election is moved to next phase.

Vote: The Vote transaction is issued by VoterParticipants and can only be executed after a StartVoting transaction like in the figure 5.1 shown. The vote is encrypted with ElGamal encryption [18] and the public key of the voting instance. Then a zero knowledge proof is executed to test the validity of the cast vote. At last step the encrypted share received from the authorities is decrypted and stored in the PrivateVoter instance.

StartTallying: The StartTallying transaction is issued by AdminParticipants. First all cast votes are checked again for their validity. Then the encrypted votes are homomorphically tallied together like explained in the preliminaries. The resulting encryption $tc(tx, tx)$ with

$$tx = \prod_{i=1}^{n} x_i \mod p \qquad ty = \prod_{i=1}^{n} y_i \mod p$$

is then stored in the PublicVoting instance and therefore visible for everyone. Note that the cast votes are stored in the PublicVoter instances, which are also

publicy readable. In this way voters can tally the votes themselves and check for the correct result.

SetDecryptor: The SetDecryptor transaction can only be issued by VoterParticipants that are decryption authorities and therefore also eligible for decryption. With this transaction a voter declares his participation in the decryption phase by creating a zero knowledge proof for the validity of the share.

StartDecrypting: The StartDecrypting transaction can be issued by any participant after the necessary threshold of decryptors is reached. First the validity of the provided shares and subkeys are tested. Then the tally is decrypted together with the provided subkeys and the result is published to the PublicVoting instance, which represents the bulletin board.

The StartDecrypting asset is very similar to the SelfDecrypt transaction, which does not write anything to the PublicVoting instance. It is used to retally the votes and check for correctness of the execution.

Chapter 6

Conclusion

We started with the intention to discover the blockchain technology for e-voting protocols. For this purpose, we analyzed and evaluated possibilities for improvement on two different protocols. We realized that openness can be traded for security, privacy and robustness with permissioned blockchains. We used Hyper-Ledger Fabric to create two such blockchains for e-voting.

We developed a new application for the Open Vote Network protocol that does not depend on cryptocurrencies and is free of transaction fees. At the same time we handled access control and made the system more secure against external attacks. The Open Vote Network is a great protocol for small boardroom elections, but has its limitations concerning voter count. In a second part, we wanted to improve a protocol that was build for large elections and thus implemented an alternative version of the Homomorphic Voting protocol proposed in [1].

In both implementations we focused on the key requirements and improved authenticity, security and privacy by defining transactions and their visibility for different participants in our network.

6.1 Future Work

There are multiple improvements that can be made for both implementations. Currently, these are proof of concepts and could be tested and improved for their performance. A frontend for both implementations can be built for real users. Different subtasks in the setup and execution of the protocol could be outsourced to different authorities, which would increase the decentralization of the protocol by reducing the trust set into AdminParticipants.

Since the implementations are only proof of concepts we stored multiple key parameters like the prime order p of the finite cyclic group G_p with standard 64 bit data type LONG. The security could be improved greatly by extending those parameters to arrays and adapt the transaction and cryptographic operations to these array data types.

More security and zero knowledge proofs for the correct execution of the protocol could be added. We only focused on the most important proofs and neglected some zero knowledge proofs regarding intermediate results and valid key generation from the authorities.

Bibliography

[1] C. Berner, "E-voting under the hood," Distributed Computing Group. ETH Zürich, Jul. 2018.

[2] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," in *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, ser. EUROCRYPT'97. Berlin, Heidelberg: Springer-Verlag, 1997, pp. 103–118.

[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Oct. 2008.

[4] F. Hao, P. Y. A. Ryan, and P. Zieliński, "Anonymous voting by two-round public discussion," *IET Information Security*, vol. 4, pp. 62–67, 2010.

[5] F. H. Patrick McCorry, Siamak Shahandashti, "A smart contract for boardroom voting with maximum voter privacy," in *Financial Cryptography*, 2017.

[6] ConsenSys, "Plcrvoting," [Online] Available: https://medium.com/metax-publication/a-walkthrough-of-plcr-voting-in-solidity-92420bd5b87c. accessed 17. March 2019.

[7] "Publicvotes," [Online] Available: https://medium.com/@DomSchiener/publicvotes-ethereum-based-voting-application-3b691488b926. accessed 17. March 2019.

[8] "Votosocial," [Online] Available: https://votosocial.github.io/. accessed 17. March 2019.

[9] "Followmyvote," [Online] Available: https://followmyvote.com/. accessed 17. March 2019.

[10] "Wavevote," [Online] Available: https://github.com/descampsk/wavevote. accessed 17. March 2019.

[11] "Polys," [Online] Available: https://docs.polys.me/technology-whitepaper. accessed 17. March 2019.

[12] A. Shamir, "How to share a secret," in *Massachusetts Institute of Technology, Cambridge, USA*, Nov. 1979.

[13] "Voatz," [Online] Available: https://voatz.com/faq.html. accessed 17. March 2019.

[14] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis, "Chvote system specification," *IACR Cryptology ePrint Archive*, vol. 2017, p. 325, 2017.

[15] V. Buterin, "A next generation smart contract and decentralized application platform," in *Ethereum White Paper*, Jul. 2015.

[16] B. L. Miguel Castro, "Practical byzantine fault tolerance," in *Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge USA*, Feb. 1999.

[17] "Introduction to hyperledger fabric docs," [Online] Available: https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html. accessed 17. March 2019.

[18] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *IEEE Transactions on Information Theory*, vol. 31. IEEE, Jul. 1985.

[19] D. Boneh, "The decisional diffie-hellman problem," in *Stanford University.*

[20] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Information Theory*, vol. 22, pp. 644–654, 1976.

[21] C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.

[22] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986.

[23] B. S. Ronald Cramer, Ivan Damgard, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Advances in Cryptology—CRYPTO, Vol. 839 of Lecture Notes in Computer Science, Springer-Verlag*, 1994.

[24] T. P. Pedersen, "A threshold cryptosystem without a trusted party," in *Lecture Notes in Computer Science*, vol. 547. Springer, Berlin, Heidelberg, May 2001.

[25] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Lecture Notes in Computer Science*, vol. 576. Springer, Berlin, Heidelberg, May 2001.

[26] J.-S. Coron, D. Lefranc, and G. Poupard, "A new baby-step giant-step algorithm and some applications to cryptanalysis," in *CHES*, 2005.