University of Zurich[UZH]

MASTER BASIS MODULE – Communication Systems Group, Prof. Dr. Burkhard Stiller

Blockchain-enabled Voting using OAuth API

Raphael Matile
Zurich, Switzerland
Student ID: 12-711-222

Supervisor: Dr. Thomas Bocek, Bruno Rodrigues, Daniel Gasteiger
Date of Submission: September 12th, 2017

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Basis Module
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

A whole new array of use cases deem applicable by using distributed ledgers. Even more, with the latest improvements of blockchains, it is now possible to run turing-complete code in a decentralized setting. Initially, the financial sector was challenged by the introduction of the digital cryptocurrency Bitcoin. However, also other systems (e.g. notary systems) have been proven useful in order to ensure a common view on certain information. In elections, decisions are required to be transparent. Common electronic voting systems use a bulletin board on a centralized server in total control of a voting authority. In contrast, this work proposes a voting system built in a distributed manner and using the consensus algorithms provided by blockchains. In particular, the outlined application is partly developed as code running in smart contracts that are deployed to the Ethereum blockchain.

Contents

Chapter 1

Introduction

Since more than two decades, voting is a highly researched topic. Challenges, not only in cryptographic areas, but also in political fields, engage research. Large-scale electronic voting can result in unforeseen and extensive societal consequences. Populations have to build trust in these systems. While this takes time, a single security issue may question all achievements in short time. However, once adopted in a broader manner, electronic voting brings in fact advantages when compared to paper-ballot voting.

Complementary to paper-ballot voting where each participant has to submit its vote on a paper ballot, electronic voting machines gained attraction due to the more efficient counting mechanisms. However, such systems have become more and more an attractive target for fraudulent activities. The presidential election in the United States of America in 2016 is among the more popular ones: According to [6], Russian military intelligence targeted a US supplier for voting software prior to the election. By 2014, the internet voting system of Estonia showed many security issues as well [16].

1.1 Motivation

Since the initial publication of the distributed ledger based cryptocurrency *Bitcoin* in 2008 [15], many different implementations of blockchains emerged. With the launch of Ethereum [1], a blockchain running so-called smart contracts in a distributed manner, a turing-complete alternative to Bitcoin and its rudimentary scripting language appeared.

Due to its architecture, a blockchain guarantees a tamper-proof history of a sequence of entries. This property is not only useful in financial applications but can also be helpful with regards to electronic voting. In general, this consensus-based data structure may ease the audit of election results. In addition, the distributed manner of blockchain applications can further increase the availability of a future voting system. The combination of these attributes may increase the adoption of blockchain-enabled voting.

1.2 Description of Work

This work aims at creating a voting protocol leveraging a blockchain for its public bulletin board. Additionally, a prototype implementing the abstract concepts of the proposed protocol is provided. However, this prototype should be seen as a work in progress, thus serving as a help to discover challenges in implementing a voting protocol on the basis of a distributed ledger.

1.3 Thesis Outline

Chapter 2 continues with a brief overview of approaches for voting with similar requirements and environments. In Chapter 3, an ideal architecture for voting on Ethereum is proposed. Outlined in Chapter 4, the voting procedure using the previously introduced system is presented. Challenges as well as limitations are then discussed in Chapter 5. Finally, this work is concluded with a brief summary in Chapter 6.

Chapter 2

Related Work

Helios[1] is a so-called open-audit voting system, relying solely on cryptographic auditing for verifying the correctness of a vote's outcome, made public in 2008. It distinguishes between two steps: A vote preparation step, in which a ballot can be casted and audited by any person wanting to take part in the election. The second step then includes the authorization of the participant as well as submitting the encrypted vote to a public bulletin board along with the voter's name. Once all voters submitted their choices to this bulletin board, the casted votes are shuffled, decrypted and counted. The coercion problem in electronic voting is assumed to be relatively irrelevant, i.e. elections are considered to be too unimportant for having coercion taking place in a substantial manner. [2]

Prêt à Voter chooses the same approach of mix-based schemes, where votes are decrypted and counted in plaintext. In contrast to *Helios*, this protocol is implemented using paper ballots and an electronic counting mechanism. Votes are encoded as a randomized list of choices, different for each voter. Therefore, the made choice does not have to be encrypted. When the vote takes place, the selection as well as the order of the choices are separated. These parts are only linked by a random number called *onion*, carrying information on how to reconstruct the choice list in the order initially presented to the voter. Since the obtained receipt of the vote does not reveal any information of how a person has voted, this scheme is to be considered coercion resistant. [14]

Voting is also being assessed in the area of blockchains. *The Open Vote Network*[2] is a self-tallying voting protocol, deployed as a smart contract on Ethereum. Voting requires two rounds, a commitment phase and the actual casting of the vote. Prior to these steps, a setup phase is conducted, performing a white-listing of authorized voters. Since this protocol is self-tallying, the last voter can perform a more informed decision on how to vote with respect to the outcome. The proposed protocol thwarts this by having an extra trusted party, always performing a dummy vote at the end of the casting. Further, this protocol relies on the property that all voters who committed to the vote, eventually will cast a vote. As soon as any of the voters does not submit a decision, the tally can not be computed anymore. To overcome this issue, the protocol introduces incentives in the form of a deposit which is returned to the voter once he has cast his decision. [10]

[1] `https://heliosvoting.org/`

[2] `https://github.com/stonecoldpat/anonymousvoting`

Chapter 3

Systems Architecture

The system proposed in this chapter uses a set of smart contracts deployed to the Ethereum blockchain prior to an election. Voting can be performed using a web application handling access to an Ethereum node which is in turn connected to the same network as the contracts were deployed to initially.

3.1 Cryptographic Procedures

This architecture relies at critical points on cryptographic operations which are briefly outlined in the next subsections.

3.1.1 Signature

Ethereum uses the SECP-256k1 Elliptic Curve Digital Signature Algorithm (ECDSA) in combination with Keccak256 as hash function for creating asymmetric keypairs. A private key sk_W is represented by a randomly chosen positive integer in the range of $[1, \texttt{secp256k1n} - 1]$ with `secp256k1n` being a specific positive number defined in [17]. The corresponding public key pk_W (the wallet's address) is obtained by using the right most 160-bits of the Keccak hash of the private key. Prior to signing, data is usually hashed [11], on Ethereum to 32 bytes of information. The procedure is then as follows: The obtained hash h_D is signed with the private key sk_W using `ECDSASIGN`, resulting in three components: r, s and v. A signature is said to be valid, if $\texttt{ECDSARECOVER}(h_D, v, r, s) = pk_W$. [17]

3.1.2 Homomorphic Enryption

Homomorphic encryption describes the ability of performing a specific operation $\oplus$ on encrypted data yielding an outcome equal to the result of having performed a related operation $\otimes$ on the plaintext, after decryption:

$$enc(m_1) \oplus enc(m_2) = enc(m_1 \otimes m_2)$$

Different homomorphic encryption systems use varying pairs of related operations: Whereas in RSA [13] both operations are multiplications, Paillier [12] uses multiplication over the encrypted messages and addition on the plaintext values. [9]

3.1.3 Zero-Knowledge Proof

Zero-knowledge proofs are guarantees that a claim is actually true, without having to reveal more than its trueness. *Interactive* zero-knowledge proofs follow a cyclic interaction between a prover and a verifier: The prover states his claim based on his secret knowledge, whereas the verifier challenges this statement calling the prover to reveal some information which is consistent to the claim made initially. The more this procedure is repeated, the more certainty about the validity of the statement is gained by the verifier. An enhanced kind of zero-knowledge proofs is called *non-interactive*, removing the need of these cyclic challenges. [9]

3.2 Votes

Transactions as well as their payload are publicly accessible in the Ethereum network. Having votes published in the blockchain which are not obfuscated in any way, allows to perform trend analysis of the undergoing election in near-realtime. In order to prohibit any such analysis, votes in the proposed system are homomorphically encrypted before submitting them to the blockchain.

The envisioned architecture currently copes only with a binary representation of choices: Votes stating a supporting decision for the question in mind are encoded as an integer *one*. On the other hand, votes opposing the question are known as being an integer *zero*. They are of type integer instead of boolean to allow a later homomorphic addition over the ciphertext. Extending this scheme to more than two choices is not hard: An indicator, as described above, can be created for each decision at hand, allowing to perform a majority vote easily.

3.3 Smart Contracts

The goal of this system is to rely as little as possible on any centralized entity interfering with the voting procedure. Besides of gaining many of the advantages of distributed systems (as increased availability) when relying on a blockchain, computations are performed multiple times, i.e. whenever a participating node tries to incorporate a broadcasted transaction to its current block, each outcome must be consistent over all performing nodes. Therefore, executing main cryptographical verifications on this infrastructure may result in an increased trust in the calculated election outcome.

3.3.1 Signature Verificator

Since the interfaces of smart contracts are publicly callable by anyone having access to the blockchain, votes could be submitted by arbitrary network participants. To restrict voting to a set of eligible voters, this contract's purpose is to check a signature passed along the vote, verifying the legality of the request. If the provided signature is invalid, the smart contract's purpose is to return an erroneous result.

3.3.2 Zero Knowledge Proof Verificator

Processed transactions in the Ethereum blockchain are accessible by everyone participating in the network and having its identifier at hand. Blockchain explorer tools as Etherscan[1] facilitate this further. To ensure voting privacy, each choice submitted to the blockchain has to be encrypted as described previously. In order to avoid the issue of having encrypted messages encipher not only a 0/1 choice but any number outside these boundaries, a zero knowledge proof has to be submitted along with the vote, ensuring that the encrypted message contains either a zero or a one. If this proof is invalid, the contract is supposed to return an erroneous result to the caller.

3.3.3 Ballot

The ballot contract is the main entrypoint of the voting system on the blockchain. Once deployed, its address can be distributed in order to let voters submit their choices.

On deployment of the smart contract, the question which has to be voted on is submitted along the contract addresses of the signature verificator as well as the Zero Knowledge Proof Verificator. Additionally, the election public key used for the homomorphic encryption is passed as well as all wallet addresses which may participate in the poll. Then, the sender of the deployment is registered as voting authority, allowing any person having the source code at hand, to initiate a new election.

To allow authorities to setup the voting procedure and any potential offline tasks related in addition to allowing to conduct a vote in a particular time range, the election must be opened resp. closed by the authorities with a corresponding call to the smart contract.

Then, voters cast their choice by sending an encrypted and signed transaction to the Ballot contract. On retrieval, the ballot checks whether the voting period has started already. If so, the sender address of the transaction is registered as having voted, ensuring that each wallet can only cast one vote and senders are not able to flood the contract with invalid votes. Then, the obtained message is forwarded to the signature verification contract, returning either a successful response if the signature of the message was valid, or a unsuccessful otherwise. In case, the signature is invalid, the voting procedure is aborted. Since the address of the sending wallet has been registered before in the ballot contract,

[1] `https://etherscan.io`

any further attempts of providing votes are denied, reducing the load on the signature verificator. If the signature was valid, the message is further forwarded from the ballot to the zero-knowledge proof verification contract. Again, the procedure terminates if the zero-knowledge proof for the message is invalid. Otherwise, the vote is registered for being counted, eventually.

Since a transaction in Ethereum may fail not only because of the invalid conditions described before, but also due to a run-out of gas or because the transaction's initiated computation reached the maximum call stack of 1024 frames, events are fired after each verification step with a corresponding state message, describing the current outcome of the initiated computation within the ballot contract.

After the voting period has ended, voting authorities may obtain the total number of submitted votes as well as each encrypted vote.

3.4 Backend

The backend component connects to the voting application with the Ethereum network and is implemented in the prototype as a Java application. This component is meant to be ran primarily by the voting authorities, however it could also be executed by a power user, wanting to submit her vote on her own infrastructure.

3.4.1 Ethereum Connection

The interaction with the Ethereum network is performed using a set of web remote procedure calls (RPC), made available by an accessible Ethereum node. In the current implementation of this research, this relies on a running Go-Ethereum node on the same machine having its RPC application programming interface (API) enabled. However, the location of the Ethereum node can be configured to run on a different node, as long as its reachable over a network.

3.4.2 Spring Boot Application

We provide a Spring Boot application[2] in order to link the voting fronted with the blockchain. It provides a websocket interface allowing callers to deploy the smart contracts to the blockchain as well as endpoints on which blockchain events may be received in near-real-time. Further it makes the frontend component accessible to requesting clients.

[2] `https://projects.spring.io/spring-boot/`

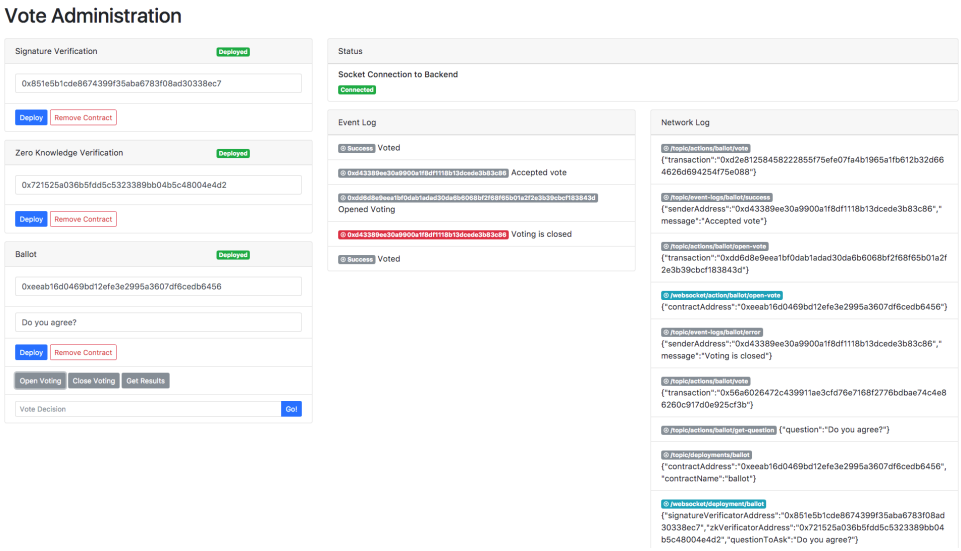

Figure 3.1: Vote Administration Interface

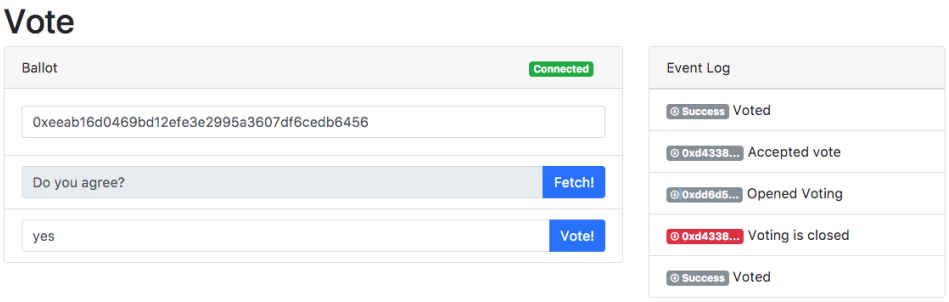

Figure 3.2: Vote Interface

3.5 Frontend

The frontend is split up in two main parts. An administration user interface as shown in Figure 3.1 forms the former, allowing the voting authority to setup a new vote, open resp. close it as well as displaying events fired on the smart contracts.

The latter, the voting interface shown in Figure 3.2 allows each voter to retrieve the question on which the voting is performed, as well as submitting a decision and receiving events of the smart contracts in order to monitor the state of the submission.

Chapter 4

Voting Procedure

The voting procedure is grouped in three sequential steps: A setup phase, in which required preconditions are met; a voting step enabling voters to cast their ballot and a final counting phase, calculating the election result.

4.1 Setup

In an ideal case, the voting authority is split among different parties, each of them running an instance of this system, making it's availability more reliant. Additionally, the smart contracts should be deployed in a semi-private network, allowing only a set of authorized nodes to incorporate transactions into blocks. Therefore, if a power user wants to run her own instance of the system, she either has to use a Ethereum node in the list of authorized nodes or having her node being authorized by the voting authority.

In Figure 4.1, three entities beside the voter are considered: A voting authority in possession of the cryptographic material used in the poll to create and having knowledge of all eligible participants, an identity provider (here labelled as *procivis*) and eventually the application proposed (labelled as *ETH Vote*).

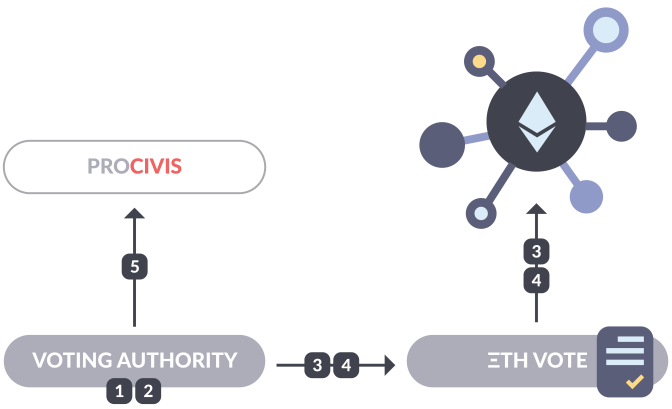

Figure 4.1: Setup Phase

Initially, the voting authority has to build a registry containing all eligible voters for the poll to create. Then, an equal number of wallets have to be set up and deposited with a high-enough amount of Ether to submit a vote. In Figure 4.1, this is represented by Step ①. Furthermore, a public-private keypair — consisting of the public key pk_E and a secret key sk_E — used for homomorphically encryption resp. decryption of votes is generated ②. Eventually, the voting authority deploys the smart contracts outlined in Chapter 3 along with all registered wallet addresses, the public key pk_E and the question to vote on to the blockchain ③. Finalizing the setup phase, the voting authority opens the vote to the public in ④.

To hinder the voting authority of relating decrypted votes to a particular voter, the identity provider is responsible of assigning wallet addresses to an authorized voter. Therefore, in ⑤ the voting authority transfers the created wallets to the identity provider.

4.2 Voting

As outlined in Figure 4.2, each voter connects to a running instance of the backend component ①. After she has been granted authorization to vote at the identity provider ②, it returns a randomly chosen address of the wallets obtained before to the voting system ③. *ETH Vote* will then fetch the question to decide on as well as the election public key pk_E, from the blockchain in ④. Then, the voter may cast his vote during ⑤. Prior to submitting the vote to the blockchain, it gets encrypted using pk_E. Correspondingly, a one-out-of-two zero-knowledge proof zk_v is generated, guaranteeing a payload of 0 resp. 1 encoded within the ciphertext. Additionally, the encrypted vote is signed using the voters wallet private key pk_w ⑥. The encrypted and signed vote is sent to the blockchain as raw transaction in ⑦, i.e. submitting the message as is.

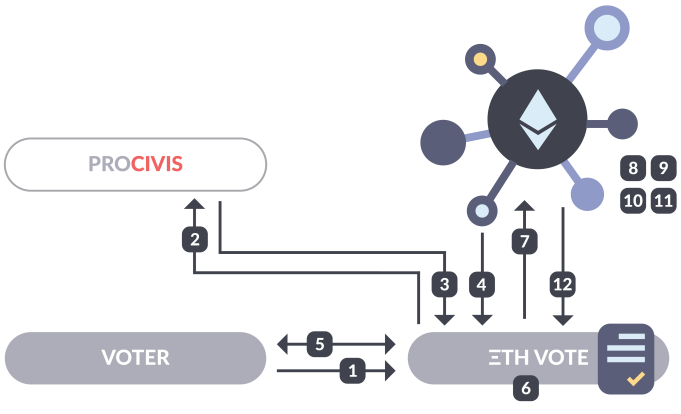

Figure 4.2: Voting Phase

On the blockchain, the ballot contract verifies that the sender's origin address is an initially registered one. Furthermore, it ensures the sender has not yet submitted any message ⑧. If so, the ballot contract forwards the received ciphertext to the signature verificator

contract. After having verified the signature, the contract sends a successful indicator to the ballot contract ⑨. Now, the ballot contract forwards the message once again, this time to the zero-knowledge verificator, which in turn audits the obtained proof to be valid ⑩. In ⑪, the ballot contract stores the vote if it obtained a successful result from the zero-knowledge verificator as well and eventually returns an "Accepted" message back to the application in ⑫. After some time, miners will have incorporated the transaction into a block, ensuring the vote to be recorded.

4.3 Counting

The counting phase is presented in Figure 4.3. It is initiated by closing the vote by the authorities using a corresponding transaction to the ballot contract ①. Then, the voting authority fetches all stored votes from the ballot contract ②.

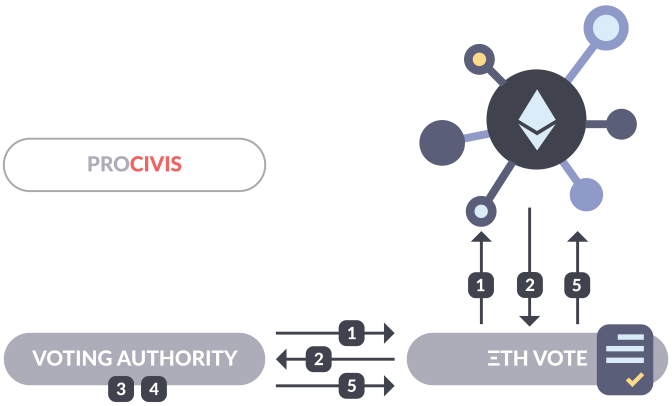

Figure 4.3: Counting Phase

On an arbitrary instance of *ETH Vote*, the tally is computed by homomorphically performing an addition on the encrypted messages. The obtained result is then decrypted using the election private key sk_E. The obtained cleartext sum represents the number of participants supporting the question posed. The amount of opposing voters is calculated by subtracting the supporting votes from the total amount of votes submitted to the ballot contract ③. In order to support the obtained result without releasing the private key sk_E to the public, allowing each and everyone a decryption of arbitrary votes, a zero-knowledge proof is generated instead, guaranteeing a correct election result ④. The election authority finally publishes the proof along with the computed results to the ballot contract ⑤, allowing every voter as well as auditors to challenge the election outcome.

Chapter 5

Evaluation

5.1 Implementation Reality on Ethereum

The following subsections challenge assumptions made in the system described in Chapters 3 and 4, and discuss their implementation feasibility with regards to the current feature set of Ethereum.

5.1.1 Keypair Generation on the Blockchain

Challenge In order to avoid having a scenario in which voting authorities may corrupt the election result or in which loosing the keypair for the homomorphic encryption to a third party takes place, it is assumed that such keys can be created on the blockchain itself. Eventually, this assumption also guarantees that the voting authorities may never come in touch with the private key, allowing them to decrypt any vote submitted.

Feasibility Current homomorphic crypto systems as RSA [13], ElGamal [8] and Paillier [12] require random numbers in their schemes. Since Ethereum is a deterministic system, requiring the same outcome for a certain transition of one state to another independently on which node it is performed, randomness is hard to achieve. Earlier attempts in the developer community leveraged blockhashes as source of randomness. Oracles, sources of information outside the blockchain, are mentioned as well, however communication is not always straightforward due to the delay between a request and its response. During this time span, some blocks in the chain might get mined and change a contract's state. Additionally, one has to trust the oracle's output, an issue to be considered less harmful if implemented by the voting authority itself. Although creating an asymmetric keypair on the blockchain could be feasible, it is not desired to do so: Data on the smart contract is available to any miner in the network as he has to compute the result of the transition function of the state before a transaction. A private key stored on a smart contract will therefore never be private but open to anyone in the network.

5.1.2 Zero-Knowledge Proof Verification

Challenge In order to ensure a submitted encrypted vote only contains a valid vote without revealing the actual decision, zero-knowledge proofs are submitted along the encrypted vote. These should be verified on the blockchain, before considering a vote being valid.

Feasibility The work presented by [10] utilizes zero-knowledge proofs as well for the very same issue. As seen in the source code[1], an implementation based on the technique outlined in [7] is possible.

5.1.3 Signature Verification

Challenge Ensuring the vote is originating from an eligible voter is heavily important when a blockchain figures as a public bulletin board. Therefore, each message received by the ballot contract must be signed using the wallet's private key. A smart contract should be able to verify, whether this signature is valid.

Feasibility Solidity provides a built in function called `ecrecover` which allows to check that a certain piece of information is signed correctly.

5.1.4 Zero Transaction Cost

Challenge To avoid having election authorities pay for participating voters in the elections, the network should allow processing transactions without paying fees.

Feasibility Ethereum allows to create private networks having only a set of white-listed nodes which are permitted to mine blocks. A network consisting of these nodes can then be configured using a custom genesis block, i.e. the first block in the blockchain. This specification can include accounts which are pre-allocated with funds. From these, a voting authority may send Ether to each wallet created during the setup phase, without having to pay real-valued fees. Since Ether is only assigned to these wallets and only an authorized set of nodes may mine transactions into blocks and therefore receive Ether as compensation, no other participant with a self-generated wallet can send transactions to the contracts deployed: Every mining node will just not accept any transaction without having received a fee for mining it.

5.1.5 Performance

When large groups of voters submit their choice to an electronic voting system on which interaction primarily happens over a website, they expect a similar performance as compared to other online services as well. As shown by Google in 2009 [5], a reduce in responsiveness directly turns out to a loss of search interactions made on their website.

[1] `https://github.com/stonecoldpat/anonymousvoting/blob/master/AnonymousVoting.sol`

Therefore, a system showing fast reactions to user input is crucial. In [10] a detailed timing analysis was performed for computation time required on the test network of Ethereum. They state an average time of $573ms$ for verifying a one-out-of-two zero-knowledge proof. This substantial amount of time in combination with the limited transaction throughput of Ethereum raises the question about being able of performing large-scale elections at all. A performance analysis on the built prototype has not been carried out, since main components of the voting procedure are not implemented but rather mocked.

5.2 Consequences

Accuracy describes the correct reflection of all voters' desire for the calculated election result. A correct implementation of the homomorphic addition process, a tamper-proof authorization module and a trustworthy voting authority may guarantee this property. However, an identity provider assigning wallets to arbitrary people corrupts the election result significantly.

Privacy characterizes the impossibility that a vote can be revealed to any third-party. Often, this includes the derived features of *Receipt-Freeness* and *Coercion Resistance*, the former being a guarantee that not even a voter can prove what he voted, whereas the latter describes the invulnerability against being forced to vote for a particular candidate [3]. Our system can be considered receipt-free with the limitation that if a malicious voting authority publishes the election private key sk_E votes can be decrypted. Having also the identity provider collaborating with the attacker, wallet addresses can be assigned to a single person, eventually revealing individuals' choices. Additionally, a voter participating from an arbitrary location having access to the Internet, may be physically threatened to a certain decision. Therefore, the proposed system architecture is not fully coercion-resistant.

Verifiability is often described in three terms: *Cast-as-intended*, *recorded-as-cast* and *counted-as-recorded*. Whereas the first describes the attribute of voters being able to independently verify that their selections were in fact correctly cast, the second describes the ability of a voter being able to review that the voting server correctly received his vote. The third kind provides auditors, as well as voters with the ability to verify that votes were received and have correctly influenced the voting result. A system holding to all three kinds of verifiability is said to be *end-to-end verifiable*.[4] Regarding the first kind of verifiability, the system could show the random parameters used for homomorphically encrypting a vote. The user then may perform the same calculations ideally ending up with the same result as the system initially showed. This may be repeated infinitely by the user, enabling him to challenge the system arbitrarily. Once satisfied, the vote is encrypted with new random parameters before actually being sent to the blockchain. [2] By looking at the transactions sent to the ballot contract on the blockchain, a voter may eventually identify the one carrying his encrypted vote, being sure that it is *recorded-as-cast*. *Counted-as-recorded* may be verified by challenging the zero-knowledge proof of the election result multiple times up to the point where a high enough probability of truthfulness is reached.

Chapter 6

Summary and Conclusions

Counting in smart contracts on Ethereum is done easily. Voting is even the beginner example in Solidity development on Remix[1], an online smart contract debugger. However, once accuracy, privacy, receipt-freeness and universal verifiability must be accounted for, protocols explode in their complexity rapidly. These challenges are not getting resolved by implementing a system on the blockchain, [3] even states that "blockchains do not address any of the fundamental problems in elections, and their use actually makes things worse". The conceptual idea of a distributed ledger removes the centralized voting authority of any election. However, such an entity is commonly already available in current offline voting systems, tasked with sending out paper ballots to the voters. Despite this fact, a publicly accessible and unchangeable track of records, storing each electronically submitted vote, increases the transparency of election results.

The described challenges and consequences, however, still remain. By focusing on a suitable trade off between a voter's privacy and the verifiability of his choice with respect to the environment the application will live in, electronic voting on blockchains may be possible in a larger scale in future years.

[1] `https://remix.ethereum.org`

Bibliography

[1] (2017). A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper.

[2] Adida, B. (2008). Helios: Web-based open-audit voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 335–348, Berkeley, CA, USA. USENIX Association.

[3] Benaloh, J., Bernhard, M., Halderman, J. A., Rivest, R. L., Ryan, P. Y. A., Stark, P. B., Teague, V., Vora, P. L., and Wallach, D. S. (2017). Public Evidence from Secret Ballots.

[4] Bibiloni, P., Escala, A., and Morillo, P. (2015). Vote validatability in mix-net-based evoting. In *Proceedings of the 5th International Conference on E-Voting and Identity - Volume 9269*, VoteID 2015, pages 92–109, New York, NY, USA. Springer-Verlag New York, Inc.

[5] Brutlag, J. (2009). Speed matters for google web search. https://research.googleblog.com/2009/06/speed-matters.html.

[6] Cole, M., Esposito, R., Biddle, S., and Grim, R. Top-secret nsa reports details russian hacking effort days before 2016 election. https://theintercept.com/2017/06/05/top-secret-nsa-report-details-russian-hacking-effort-days-before-2016-election/.

[7] Cramer, R., Damgård, I., and Schoenmakers, B. (1994). Proofs of partial knowledge and simplified design of witness hiding protocols. *Advances in Cryptology – CRYPTO*, 839:174–187.

[8] ElGamal, T. (1985). *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, pages 10–18. Springer Berlin Heidelberg, Berlin, Heidelberg.

[9] Jonker, H., Mauw, S., and Pang, J. (2013). Privacy and verifiability in voting systems.

[10] McCorry, P., Shahandashti, S. F., and Hao, F. (2017). A smart contract for board-room voting with maximum voter privacy. In *Financial Cryptography and Data Security*.

[11] National Institute of Standards and Technology (2009). Digital Signature Standard (DSS). *Fips Pub 186-4*, (July):1–119.

[12] Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg. Springer-Verlag.

[13] Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126.

[14] Ryan, P. Y. A. (2008). Prêt à Voter with Paillier encryption. *Mathematical and Computer Modelling*, 48(9-10):1646–1662.

[15] Satoshi, N. (2008). Bitcoin: A peer-to-peer electronic cash system. Technical report.

[16] Springall, D., Finkenauer, T., Durumeric, Z., Kitcat, J., Hursti, H., MacAlpine, M., and Halderman, J. A. (2014). Security analysis of the estonian internet voting system. `https://jhalderm.com/pub/papers/ivoting-ccs14.pdf`.

[17] Wood, G. (2017). Ethereum: a secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, EIP-150 RE.

Glossary

Ether "The primary internal cryptographic token of the Ethereum network. Ether is used to pay transaction and computation fees for Ethereum transactions." [1]

Gas and Gas Limit "Every transaction has a specific amount of gas associated with it: gasLimit. This is the amount of gas which is implicitly purchased from the sender's account balance. The purchase happens at the according gasPrice, also specified in the transaction. The transaction is considered invalid if the account balance cannot support such a purchase. It is named gas limit since any unused gas at the end of the transaction is refunded (at the same rate of purchase) to the sender's account. Gas does not exist outside of the execution of a transaction." [17]

List of Figures

Appendix A

Installation Guidelines

In order to run the prototype available on GitHub[1], perform the following steps:

1. Fetch the repository from GitHub.
2. Run `git submodule init && git submodule update` in order to also fetch the frontend included as submodule.
3. Install and start the Go-Ethereum client. Make sure, you enable the RPC API: `geth -rpcapi personal,db,eth,net,web3 -rpc -testnet`.
4. Verify that Geth is running on `http://127.0.0.1:8545`.
5. Start the Spring boot application by running `mvn spring-boot:run`
6. You may want to make sure that enough Ether is present on the test wallet distributed along with the application. Its wallet address is `0xd43389ee30a9900a1f8df1118b13dcede3b83c86`. Use the throttled faucet[2] to send som Ethers to it, if you get deployment errors.
7. You should now be ready to visit `http://localhost:8080/admin/index.html` for the Admin Interface as well as `http://localhost:8080/voter/index.html` for the Voting interface.

Note that you have to deploy the `Signature Verification` and `Zero Knowledge Verification` contract before you may deploy the `Ballot` contract. In order to achieve this, click on `Deploy` for both contracts and wait until their address in the testnet are shown. Then, you have to specify a voting question in `Ballot` prior to hitting `Deploy` on its corresponding button. Once the `Ballot` contract has received its address as well, you can open the voting, publish a vote and eventually close the vote again. Further note, that you can only submit a vote once, since currently the address of the wallet shipped along the application is used as sender and only one vote by sender address is allowed.

[1] https://github.com/rmatil/uzh-ethereum-vote

[2] `http://ipfs.b9lab.com:8080/ipfs/QmQsrJAyf3vv7otNoiYDMzcVN31o62Hq22yr8df4LgdFmr/throttled_faucet.html`